UCLouvain

École polytechnique de Louvain

A Framework for Engineering Gesture-Based User Interfaces Using the Leap Motion Controller

Development and Applications

Author: **Arthur SLUŸTERS**
Supervisor: **Jean VANDERDONCKT**
Readers: **Suzanne KIEFFER, Sébastien LAMBOT**
Academic year 2019–2020
Master [120] in Computer Science and Engineering

Abstract

The continuous improvement of gesture-sensing technologies by sensors, wearable devices, and computer vision is fostering a shift towards more natural user interfaces taking benefit of these technologies. After mastering 2D gesture recognition on contact-based interactive surfaces, researchers and practitioners are turning their focus of interest to contact-less gestures performed in mid-air interaction spaces. The Leap Motion Controller has become a ubiquitous device for capturing contact-less mid-air gestures that led to numerous projects and applications. Since the Leap Motion comes with an Application Programming Interface, the development of gesture-based user interfaces is primarily achieved by hand programming in an opportunistic way, thus impeding maintainability, portability, and compatibility. To satisfy these software quality properties, we conduct a target literature review in Leap Motion gesture recognition to develop a framework for engineering gesture-based user interfaces captured by a Leap Motion that preserves these properties. Its generic and flexible software architecture accommodates several configurations for gesture recognition, testing, and benchmarking of gesture recognizers, as well as the deployment of interactive applications with Leap Motion gesture-based user interfaces. We illustrate the development process on DICOM Viewer, an application for browsing medical images by gestures. Then, we perform a benchmarking of implemented gesture recognizers. Then, we deploy a large-size, real-world application in the context of the Large User Interface (LUI), a project launched by MIT Media Lab. Finally, we present a plan for evaluating the framework and an application deployed based on it and we discuss the benefits and shortcomings of this work before suggesting some future avenues.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my thesis supervisor, Prof. Jean Vanderdonckt, for his precious guidance. This thesis would not have been possible without his insightful advice and feedback.

I would also like to thank Prof. Radu-Daniel Vatavu from University Stefan cel Mare of Suceava for taking the time to advise me during his scientific stay in Belgium and for valuable help on the gesture recognizers that he (co-)developed.

Thank you to the two readers of my thesis, Prof. Suzanne Kieffer (SSH/ESPO/ COMU and SSH/ILC/PCOM) and Prof. Sébastien Lambot (SST/AGRO and SST/ELI/ELIE).

Many thanks to Mehdi Ousmer and Nathan Magrofuoco for their much appreciated help and support in some crucial parts of my thesis. Also a special thank you to Prof. Paolo Roselli for his fundamental support in geometry-based gesture recognition.

A huge thank you to Bastien Nothomb, for his collaboration on the framework. It was a real pleasure working with him and I wish him the best for the future!

A special thank you to Vik Parthiban, the main developer of the LUI project, managed by Prof. Pattie Maes of MIT, who provided the functional core of the LUI project for which we develop a complete gesture user interface.

Finally, I would like to express my deep and sincere gratitude to my friends and family for putting up with my stress and for their support throughout this year. I could not have done it without them.

Contents

1 Introduction **1**
- 1.1 Context of the Problem . . . 1
- 1.2 Mission Statement . . . 3
  - 1.2.1 Statement . . . 3
  - 1.2.2 Working Hypotheses . . . 3
  - 1.2.3 Research Method . . . 3
- 1.3 Overview of the Thesis . . . 4

2 Related Work **5**
- 2.1 The Leap Motion Controller . . . 5
- 2.2 LMC-based Gesture Recognition: Research and Development . . . 6
- 2.3 Template-Based Gesture Recognizers . . . 8
- 2.4 LMC Gesture Sets . . . 10
- 2.5 Summary . . . 11

3 Framework Development **13**
- 3.1 Requirements . . . 13
- 3.2 Software Architecture . . . 14
  - 3.2.1 Overview . . . 14
  - 3.2.2 Frozen Spots . . . 15
  - 3.2.3 Hot Spots . . . 17
  - 3.2.4 Configuration File . . . 21
- 3.3 Use-Case: the DICOM Viewer . . . 22
  - 3.3.1 Overview . . . 22
  - 3.3.2 Gesture Set . . . 23
  - 3.3.3 Configuration and Efficacy of the Framework . . . 23
- 3.4 Conclusion . . . 24

4 Benchmarking of Recognizers **25**
- 4.1 Benchmark . . . 25
  - 4.1.1 Benchmarking Tool . . . 25
  - 4.1.2 Recognizers . . . 25
  - 4.1.3 Experiment . . . 27
- 4.2 Results . . . 28
  - 4.2.1 Recognition Rate . . . 28
  - 4.2.2 Execution Time . . . 30
- 4.3 Conclusion . . . 31

5 The Large User Interface Project **33**
5.1 Project Overview . . . 33
5.1.1 Rationale . . . 33
5.1.2 Current State . . . 34
5.2 Designing an Appropriate Gesture Set . . . 34
5.2.1 Preliminary Gesture Elicitation Studies . . . 34
5.2.2 New Gesture Elicitation Study . . . 37
5.2.3 Results . . . 38
5.2.4 Choice of Gestures . . . 40
5.2.5 Gesture Taxonomy . . . 40
5.3 Finding the Most Appropriate Gesture Recognizers . . . 42
5.3.1 Procedure . . . 42
5.3.2 Results . . . 43
5.4 Implementation . . . 45
5.4.1 Framework Configuration . . . 45
5.4.2 Modifications to the Project . . . 46
5.5 Conclusion . . . 48

6 Evaluation **49**
6.1 At the Level of a Gesture-Based User Interface With the Framework . . . 49
6.1.1 Plan for Evaluating User Satisfaction . . . 50
6.2 At the Level of the Framework Itself . . . 52
6.2.1 Does the Framework Meet its Requirements? . . . 52
6.2.2 Qualitative Evaluation of the Framework . . . 53

7 Final Conclusion **55**
7.1 Overview of the Contributions . . . 55
7.2 Future Work . . . 56

A Work Distribution **77**

B Preliminary Gesture Elicitation Studies **79**
B.1 General Information About the Experiments . . . 79
B.1.1 Participants . . . 79
B.1.2 Apparatus . . . 79
B.1.3 Task . . . 80
B.1.4 Measures . . . 80
B.1.5 Taxonomy . . . 80
B.2 Photo-Browsing . . . 81
B.2.1 Experiment . . . 81
B.2.2 Results . . . 82
B.3 Video-Browsing . . . 84
B.3.1 Experiment . . . 84
B.3.2 Results . . . 85
B.4 What Can We Conclude From Both Experiments? . . . 87
B.4.1 Agreement Rate, Thinking Time and Goodness of Fit . . . 87
B.4.2 Taxonomy of Gestures . . . 87
B.4.3 Guidelines for Future Elicitation Studies . . . 88

C Recording and Playback Tool **93**

D New Elicitation Study **95**

E LUI Gesture Set **97**

F Code of the Software **99**
F.1 Framework . . . 99
F.1.1 Frozen Spots . . . 99
F.1.2 Hot Spots . . . 118
F.1.3 Configuration Files . . . 123
F.2 Sensors . . . 137
F.2.1 Leap Motion Controller . . . 137
F.3 Datasets . . . 140
F.3.1 Static Gestures . . . 140
F.3.2 Dynamic Gestures . . . 143
F.4 Classifiers . . . 148
F.4.1 GPSDa . . . 148
F.4.2 P^3+ . . . 151
F.5 Analyzers . . . 159
F.5.1 Basic Analyzer . . . 159
F.6 Segmenters . . . 161
F.6.1 Left-Hand Segmenter . . . 161
F.6.2 Zoning Segmenter . . . 162
F.6.3 Window Segmenter . . . 164
F.7 Recognizers . . . 167
F.7.1 3 cent . . . 167
F.7.2 P^3 . . . 171
F.7.3 P^3+ . . . 177
F.7.4 P^3+X . . . 184
F.7.5 Q^3 . . . 192
F.7.6 Jackknife . . . 200
F.8 Large User Interface . . . 203
F.8.1 Welcome Screen . . . 203
F.8.2 App Menu . . . 209
F.8.3 Videos App . . . 212
F.8.4 Photos App . . . 228
F.8.5 PDFs App . . . 244

Chapter 1

Introduction

1.1 Context of the Problem

For decades, people have been exposed to futuristic user interfaces (UIs), mostly through Sci-Fi movies and series. They have seen their favorite characters ditch the classic keyboard-mouse-screen trio in favor of a more high-tech combination of gestures, wall screens, and holograms, with gesture interaction as the primary modality of interaction [48,56].

The continuous improvement of motion-sensing technologies by sensor input and/or computer vision [27,68] is fostering a shift towards more natural UIs taking benefit of these technologies. After mastering 2D gesture recognition [110,111,119,122] on touch-enabled, surface computing [133], researchers and practitioners are turning their focus of interest to 3D motion gestures performed in mid-air interaction spaces [136]. For example, to establish some continuum from 2D to 3D gestures [10], a "touch+air" gesture [12] initiates a 2D gesture captured on a surface and terminates it with a mid-air gesture, or vice versa in "air+touch" [26].

Although long perceived as a distant dream, motion-tracking technology has become more affordable at the beginning of this millennium. As a result, it has already been used in various mainstream applications, such as the Nintendo Wii, released in 2006, which came with a literally "game-changing" controller that combined accelerometers with an infrared sensor to determine its position in space. In 2010, Microsoft released Kinect, a device composed of depth sensors and cameras that captured the position of various body parts. Coupled with an Xbox 360, it provided a gesture-based gaming experience to the consumers

Released in 2012, the Leap Motion Controller (LMC) focused on precisely tracking the positions of hands and their fingers, to navigate through computer interfaces. Other motion-sensing devices include smart rings [42] and wired gloves [40], but their impact on the consumer market was not as significant as the previous examples. Since its inception, the Leap Motion Controller has initiated a very large amount of projects in mid-gesture recognition and interactive applications equipped with a gesture-based UI [58].

The LMC captures raw gestures from its sensor and makes them available through a Application Programming Interface (API) available in various languages, such as C++, Java, Python, and JavaScript [70]. Therefore, developing a gesture-based UI based on the LMC requires extensive, manual programming [108] in one or many of these languages [85], which are challenging to master for advanced UIs [101]. It turns out that most projects and applications, if not all, have followed different hand-crafted, fit-to-purpose, opportunistic practices to develop their UI. This manual practice largely impediments three fundamental ISO/IEC 205010 software quality properties [50]: compatibility, maintainability, and portability, along with some of their sub-properties:

1. *Compatibility*: "degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions while sharing the same hardware or software environment". This characteristic is composed of the following sub-characteristics [50]:

    - *Co-existence*: "degree to which a product can perform its required functions efficiently while sharing a common environment and resources with other products, without detrimental impact on any other product". For the LMC, there is no common environment raising the level of abstraction with respect to gesture raw data to leverage the development of corresponding applications.
    - *Interoperability*: "degree to which two or more systems, products or components can exchange information and use the information that has been exchanged". For the LMC, various projects and applications partially or totally use all data provided by the LMC API and store them in different formats that are not interoperable.

2. *Maintainability*: "degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements". This characteristic is composed of some of the following sub-characteristics [50]:

    - *Modularity*: "degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components". For the LMC, it is unsure to what extent the software architecture implemented is structured into adequate layers and modules to guarantee this sub-property.
    - *Reusability*: "degree to which an asset can be used in more than one system, or in building other assets". Due to inconsistencies, existing LMC implementations are not reusable for another application. In our review, we did not identify any architecture that is reused from one project to another.
    - *Modifiability*: "degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality". It is unclear how existing LMC implementations could be subject to any modification, if any, that would support new gesture recognition.
    - *Testability*: "degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met". Beyond the testability of existing LMC-based implementations, most studies report their own results for gesture recognition, with little or no comparison to others.

3. *Portability*: "degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another". This characteristic is composed of the following sub-characteristics [50]:

    - *Adaptability*: "degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments". LMC-based applications are often tied to a particular usage environment and are rarely adaptable to another, such as for another context of use (*i.e., different users, platforms, and environments* [16]).
    - *Installability*: "degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment". Part perhaps for Java and JavaScript which are multi-platform programming languages,

LMC-based implementations can only be installed in some particular environment, such as C# for MS Windows and Objective-C for MacOS. These setups cannot be transferred to another platform.

- *Replaceability*: "degree to which a product can replace another specified software product for the same purpose in the same environment". LMC-based applications usually implement one gesture recognition technique that cannot be replaced by another one. Even more, multiple recognition techniques cannot be combined.

1.2 Mission Statement

1.2.1 Statement

In order to address the aforementioned shortcomings affecting compatibility, maintainability, and portability of LMC-based UIs, the central objective of this master thesis is to **develop a framework for engineering gesture-based UIs of interactive application using the LMC**. The framework should be generic enough to explicitly address the three software quality properties and their corresponding sub-properties.

1.2.2 Working Hypotheses

During this thesis, we assume the following working hypotheses:

H1. *Device and platform specifications*: the framework should facilitate the deployment of gesture-based UIs based on the LMC [70] only, for mid-air gesture interaction [58], and should be employed on multiple computing platforms, and not only one with a specific operating system. We pose the hypothesis of working in JavaScript. Other similar devices are not considered.

H2. *Environment specifications*: the LMC-based UI will be used in an in-door, stable environment with the appropriate lighting conditions. Non-wearable, contact-less input sensing devices, such as the LMC, are better suited to the environment than wearable, contact-based devices, such as an interactive surface.

H3. *Target audience*: the framework is aimed at facilitating the development life cycle of LMC-based UIs for the developer and the designer.

H4. *End-users' aspects are out of scope*: although usability is also specified in ISO/IEC5010 as another important quality property, it will not be addressed in this thesis in computer science. Consequently, no particular assumption is made about end-users, their profiles, or their abilities. Other ISO properties such as security, functionality are also beyond the scope of this thesis.

1.2.3 Research Method

Our research method is structured according to the software development process, which is usually divided into six phases according to the following definitions inspired by the IEEE Standard Glossary of Software Engineering Terminology [49]):

1. *Requirements analysis*: studying user needs to arrive at a definition of system or software requirements.

2. *Design*: creating and documenting designs for the architecture, the software components, the interfaces, and the data.
3. *Implementation*: translating the design into a software component.
4. *Testing*: checking whether or not the system is meeting our expectations.
5. *Deployment*: delivering the product to the customer.
6. *Maintenance*: modifying the system after the delivery to correct faults, improve performance or other attributes, or adapt to a changing environment.

In the context of this thesis, we will mostly consider the first three steps (from requirements to implementation) in the development of the framework, the fourth step (testing) through running a benchmarking in LMC-based gesture recognition and the fifth step (deployment) through deploying two LMC-based applications: a small-size DICOM viewer and a large-size application for manipulating multimedia contents on wall screens in the context of the Large User Interface (LUI) project [89] managed by The MIT Media Lab[1].

1.3 Overview of the Thesis

To reach the central objective assigned to this thesis, the manuscript is structured as follows.

Chapter 2 explores previous work related to this objective by describing the LMC, by conducting a targeted literature review in the area of LMC-based projects, applications, and recognition techniques and by performing a comparative analysis of gesture recognizers that are relevant for the context.

Chapter 3 covers the first three stages: the requirements engineering, the design, and the implementation phases for the framework for engineering LMC-based UIs. In particular, the pipeline software architecture, its underlying unified form for gesture representation, and the implemented gesture recognizers will be described. It will end up illustrating the development process for DICOM Viewer, a small-size application for browsing medical images by gesture, in order to address the phase of deployment. This application gives rise to a first framework instantiation.

Chapter 4 addresses the phases of testing and deployment by setting up the framework in a configuration useful for performing a benchmarking of implemented recognizers on two challenging Leap Motion datasets. This application gives rise to a second framework instantiation.

Chapter 5 also addresses the phase of deployment for developing a gesture-based UI for the LUI project for browsing multimedia objects on wall screens. This application gives rise to a third framework instantiation.

Chapter 6 briefly discusses an experimental plan for evaluating the framework and an application deployed thanks to this framework, such as the LUI project. It also summarizes the benefits and shortcomings of the work in light of the three properties: compatibility, maintainability, and portability.

Finally, Chapter 7 overviews the contributions of this thesis and suggests some future avenues to this work, given its existing shortcomings.

[1]This work is part of a larger collaboration project between MIT and UCLouvain in the context of an MIT-UCL MISTI Fund "Large User Interface" project (see https://www.media.mit.edu/projects/large-user-interface-with-gesture-and-voice-feedback/overview/) lead by Prof. Pattie Maes on the MIT side and by Prof. Jean Vanderdonckt on the UCLouvain side, under contract #200278.

Chapter 2

Related Work

After introducing the Leap Motion Controller (LMC) and presenting its capabilities in Section 2.1, this chapter is structured as follows from the most general to the most specific: Section 2.2 overviews general research and development performed based on the LMC, Section 2.3 explains some template-based gesture recognizers that are specifically adapted to stroke gesture recognition, and Section 2.4 describes the typical gestures and gesture sets used with the LMC in existing projects and applications.

2.1 The Leap Motion Controller

The Leap Motion Controller (LMC) is a small ($80\times30\times13mm$) device that connects to a computer via its USB port. It is composed of two near-infrared cameras which allow it to track motion in a cone-shaped zone extending about $60cm$ from the base of the device (see Figure 2.1). It provides motion tracking data in the form of a skeleton. According to [128], it manages sub-millimeter ($0.2mm$) accuracy for static gestures, and falls to $1.2mm$ for dynamic gestures. The manufacturer advertises a typical frame rate of about $120Hz$.

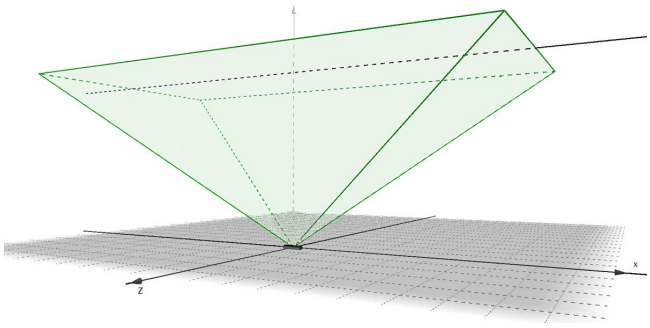


Figure 2.1: Field of view of the Leap Motion Controller (adapted from [69]).

The Leap Motion Software Development Kit (SDK) is available for many programming languages, including Javascript, Python, and C++ [85]. However, only the last version of their LeapC API, as well as their integration for Unity and Unreal Engine built on this API, are still actively supported. The API transforms raw data acquired from the hand gesture into Frame objects. Several properties exist for each frame [101]: the palm position, its normal form and its velocity (Figure 2.2, left), the hand direction, the position, the direction, and the speed of the five fingertips (*i.e.,* the thumb, the index, the middle, the ring, and the pinky), and the arm direction (Figure 2.2, right).

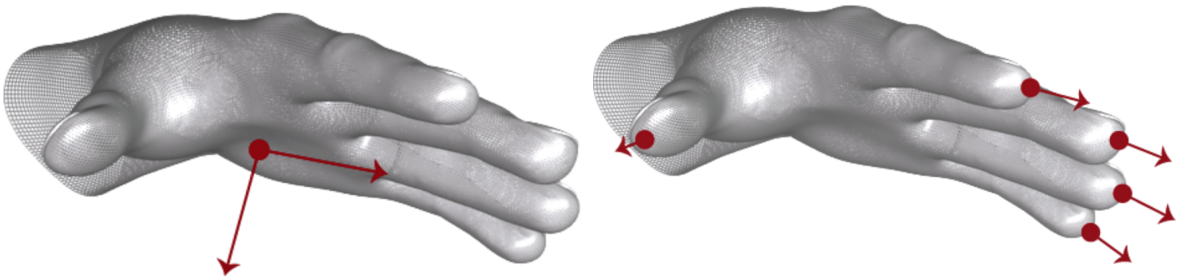


Figure 2.2: Leap Motion data: palm tracking (left), fingertips (right) [70].

There are two main types of LMC-based hand gestures: *static gestures*, where the pose of the hand(s) entirely conveys the meaning of the gesture; and *dynamic gestures*, which consist of a sequence of hand poses in time. Continuous gesture recognition consists of both gesture segmentation (*i.e.,* extracting a gesture from a continuous stream of data) and gesture recognition. Some recognition algorithms work directly from raw sensor data (*e.g.,* images), whereas other work from skeleton data (*i.e.,* a series of body joints and their position, speed, etc.). We do not consider here methods for extracting a skeleton from raw sensor data. Because of its design, the Leap Motion Controller suffers from a few issues:

- Accuracy decreases when parts of the hand are hidden from the sensor. As such, some intricate hand poses might not be picked up correctly, such as in self-occlusion.
- The interaction zone is quite small because of the sensor's limited field of view. The LMC is thus mostly limited to hand gestures, as opposed to other devices such as the Kinect, but which are much more sophisticated and less specifically tailored for hand recognition.
- The controller might sometimes mistake one hand for another. For instance, it might detect two left hands instead of separating the left hand from the right hand.

2.2 LMC-based Gesture Recognition: Research and Development

We conducted a Targeted Literature Review (TLR), which is a non-systematic, in-depth, and informative literature review, that keeps only the references maximizing rigorousness and relevance while minimizing selection bias [67]. The relevant references obtained may be more limited than in a Systematic Literature Review (SLR) [54], but stems for a knowledgeable selection of high-quality, easy-to-identify references on 2D stroke gesture recognition, as opposed to an all-encompassing list of somewhat relevant references. This applies to LMC gesture recognition, where the literature is very abundant [39,44].

3D gesture recognition is a huge field of research and development given the proliferation of devices, wearable or not, the human limbs considered for gesturing, and the techniques used for recognition. We refer to the literature for a more extensive survey, such as to [27] for 3D hand gesture recognition (a complex three-step process involving hand detection, hand tracking, and recognition [93]) and more recently to [136] for a systematic literature review. They found that the main techniques used between 2016 and 2018 are: general purpose toolkits (*e.g.,* [129]), Artificial Neural Networks (ANN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), k-Nearest Neighbor (k-NN [37], *e.g.,* [125]), decision tree classifier, Support Vector Machine (SVM), Hidden Markov Model (HMM), Dynamic time warping (DTW, *e.g.,* [111]), Naive Bayes classifier, Linear discriminant analysis (LDA, *e.g.,* [97]), Principal

Component Analysis (PCA), Deep Learning (*e.g.,* DeepGRU [77]), etc. A common pattern identified in this portion of the literature is the overfitting in the data sets for artificial intelligence and machine learning algorithms: they exhibit excellent accuracy (usually superior to 90%) and performance (usually inferior to 100 msec., a usual threshold for system response time [87]), but they are entirely optimized for a given dataset, thus leaving little or no space for keeping the same performance if the dataset is changed. This occurs when samples and/or classes need to be modified, thus leading to a re-training of the algorithm, although this is now achievable in almost real-time. For example, when a system-defined gesture should be replaced by a user-defined gesture in a dataset, a re-training may be required.

Based on this TLR, we identified a series of significant gesture recognition techniques used in research and development that are exploiting the LMC.

Neto *et al.* [86] present two ANNs to spot and classify static hand gestures in a continuous stream of data from wired gloves. The two ANNs are arranged in series, where the first one focuses on spotting communicative gestures, while the second one is trained to detect transition (non-communicative) gestures. The authors claim that their method achieves high accuracy with a short training time of about 10 to 15 minutes.

Marin *et al.* [80] combine features extracted from a Microsoft Kinect's depth map with features provided by a Leap Motion Controller to achieve higher accuracy than with either of these 2 devices alone. They use a multi-class SVM classifier to classify extracted gestures and manage to reach 91.3% accuracy. They improved their initial solution by using data from the Leap Motion Controller to help extract data from the Kinect device [81]. This approach increases performance as well as accuracy, which now reaches 96.5%. The authors also measure the impact of replacing the multi-class SVM by a Random Forest. It results in faster training times at the cost of slightly worse accuracy (94.7%). The authors claim that their system can provide real-time performance, even on low-end computers.

Bhuyan *et al.* [9] propose a vision-based approach for sign language recognition. They highlight the challenges of gesture segmentation in a continuous data stream. These challenges include handling coarticulation (the fact that a gesture is influenced by those preceding and following it) and transition gestures (non-gestural movements between gestures). Their solution uses Conditional Random Fields (CRFs) instead of HMMs because are better at detecting gestures when there is coarticulation.

Avola *et al.* [8] use a stack of four RNNs to recognize sign language from skeleton data acquired using a Leap Motion Controller. They achieve 96% accuracy on a custom dataset of 12 dynamic and 18 static gestures.

Kratz & Back [59] show that it is possible to accurately determine the execution phases (*i.e.,* the start, middle, and end) of pre-segmented gestures recorded with a smartphone's IMU (Inertial Measurement Unit, a combination of a gyroscope, accelerometer, and magnetometer). They used a multi-class SVM, but suggest that it may be possible to determine the same phases in unsegmented data using HMMs.

Alon *et al.* [4] propose a framework to handle spatial (work out the position of the hand in an image) and temporal (determine the start and end of a gesture) gesture segmentation, as well as gesture recognition in a continuous video sequence. Instead of requiring a pause between two consecutive gestures, their framework determines gesture boundaries by finding intervals of the input sequence that seem to match with one of the gesture class. The authors show two possible applications of their framework: real-time recognition of digit gestures, and a system that helps users find a specific sign within a video containing continuous signing in ASL.

Krupka *et al.* [65] show the basics of what later became Microsoft Prague [95]. It is an SDK that developers can use to add hand-gesture interaction to any application. Gestures are

represented as finite-state machines (FSMs), whose transitions can be either hand poses, hand motions, or other gestures. Hand poses are described as sets of constraints, while hand motions are a sequence of 2D "motion building blocks" that can be attached to specific parts of the hand. A gesture is recognized if the user performs the action sequence represented by its FSM. There is no need for training data, as gestures are entirely defined using their description language. Using this method, they achieve 96% recognition rate on a set of 12 gestures. At the moment, their SDK runs exclusively on Windows 10 and supports only three depth cameras: the Kinect v2 as well as the F200 and SR300 Intel RealSense cameras. It is unclear whether they will ever update it to support cheaper devices such as the LMC.

2.3 Template-Based Gesture Recognizers

Several algorithms recognize 2D stroke gestures [138] by adopting a *template matching* [14]: they compare a *candidate gesture* against pre-recorded *template gestures* gathered by *gesture classes* in a *training set* that was used to train the recognizer beforehand. This assumes calculating a (dis-)similarity distance between the candidate gesture and template classes.

We consider these recognizers in this work because they are quick and easy to implement, while still delivering a high enough recognition rate, albeit usually lower than what can be achieved with more advanced techniques. Although they were initially reserved for 2D unistroke gestures, we will consider them later on for generalizing them to 3D LMC-based gestures.

GrandMa, Rubine's recognizer [97], creates a vector of 13 geometric features for each template and candidate, which is classified with respect to gesture classes by a linear evaluation. This offers a high recognition accuracy for a low computational cost, manual opportunistic programming is no longer required, and its understanding is simple and facilitated by a geometric interpretation. These three advantages form a common motto for several other recognizers.

Wobbrock *et al.* present their **$1 recognizer** [133], a 2D unistroke gesture recognizer specifically designed to be used in prototyping environments. It can fit in about 100 lines of code and can recognize gestures with as few as one template per gesture class. Candidate gestures first go through a four steps preprocessing: (1) spatial resampling; (2) approximate angular alignment; (3) non-uniform scaling; and (4) translation. The algorithm then iteratively searches for the optimal angle between the candidate gesture and each template using Golden Section Search, and the template with the lowest Euclidean distance is returned. Due to its simplicity, it does have a few caveats. It is limited to 2D unistroke gestures and cannot recognize 1D gestures (*e.g.,* lines) because of its non-uniform scaling. Furthermore, it is rotation, scale, aspect-ratio, space, and time-independent, which may cause problems in some situations, such as when distinguishing ovals and circles, or gesture execution speed is important.

Anthony & Wobbrock's **$N** and **$N-protractor** recognizers [5,6] address most of $1's weaknesses. They add support for 2D multistroke gestures by connecting the strokes of a gesture to form a single unistroke gesture. Because a gesture's strokes may be performed in different orders and directions, both algorithms automatically generate all the permutations of a template gesture's strokes. This approach allows the designer to provide only one template for each multistroke gestures but significantly increases time and space complexity. The impact on time complexity is partially mitigated by the simple and fast comparison algorithm, as well as by 2 optimizations proposed by the authors, which can reduce the number of comparisons by up to 90% without any negative impact on accuracy. Another improvement compared to $1 is the support for 1D gestures (*e.g.,* lines): 1D gestures are automatically recognized and scaled uniformly to preserve aspect ratio, while 2D gestures are still scaled non-uniformly.

$N-protractor replaces the iterative search for the best angular alignment of $N by an approach based on inverse cosine distances. This significantly increases its recognition speed compared to $N, with a negligible impact on its accuracy. Both recognizers are scale and position-independent and suffer from a high drop in performance when the number of templates and the number of strokes per template increases. Finally, because both algorithms connect strokes to form a single unistroke gesture, there may be collisions between symbols (*e.g.,* "z" and "=").

Vatavu *et al.*'s **$P recognizer** [121] try to overcome $N and $N-protractor's memory explosion problem. Instead of considering a gesture as a sequence of strokes, it represents them as a "time-free cloud of points". This approach considerably reduces memory usage, as the recognizer does not need to store each permutation of the multistroke gestures. However, it loses the ability to distinguish gestures according to their direction (*e.g.,* it cannot distinguish "swipe up" from "swipe down").

The $Q recognizer [122] improves $P with several algorithmic revisions: the classification of multi-stroke gestures is faster when the size of the training set is large. Aligned with the accuracy of $P, $Q is announced as the fastest recognizer of the $-family. Other variants to the $-family also exist, such as Quick$ [94], 1F [117], Shark [64], Shark2 [63], and 1¢ [45].

More recently, G-Gene [19] declaratively models stroke gestures like DG3 [35] for recognizing a stroke in real-time, as a whole or for its parts, with an accuracy comparable to offline methods already after a few segments of the stroke.

Vanderdonckt *et al.* [115] present **!FTL** as an alternative to $1 and $P. It is a vector-based 2D multistroke gesture recognizer that converts resampled gesture points into a series of pairs of vectors called "shapes" and computes the Local Shape Distance (LSD) between each pair of shapes in sequence to assess how two gestures are dissimilar. Its accuracy is comparable to $P, but four times faster. !FTL has not been compared to $Q in terms of speed, though.

Both Kratz and Rohs' **$3** and **Protractor3D** recognizers [60,61] attempt to recognize gestures regardless of their orientation, although they use different techniques to align the performed gesture with the templates. The former rotates input gestures independently from the templates while the latter finds the rotation of the input gesture that minimizes the Euclidean distance to each template. Both recognizers then return the template with the lowest Euclidean distance to the input gesture.

Caputo *et al.*'s **3 cent recognizer** [18] is based on the same idea of comparing sampled trajectories as Kratz and Rohs' recognizers, but it assumes that gestures are always performed in the same orientation. According to the authors, this allows their recognizer to be significantly faster and more accurate. However, it requires each gesture to be defined precisely, with a fixed direction and orientation. They find that classification stays very accurate with as few as 10 sampling points and that a gesture can be recognized before its completion, as accuracy stays about the same even if classification is performed on only the first 60% of the gesture.

Taranta *et al.*'s **Jackknife** [111] is a general-purpose algorithm that can recognize dynamic gestures from continuous streams of data produced by a variety of devices, including touchscreens and 3D sensors. In the spirit of the $-family recognizers, it manages high recognition rates with limited training. However, unlike them, it uses DTW (Dynamic Time Warping) to measure the distance between two templates. It supports unsegmented data by using a sliding-window approach, combined with an automatically determined rejection threshold to avoid false positives.

Celebi *et al.* [24] achieve a 36% gain in accuracy compared to classical Dynamic Time Warping by using a distance function that assigns a different weight to each body joint, depending on its relevance in the recognition of this gesture class.

Vatavu *et al.* [118] study the impact of gesture dimensionality (*i.e.,* the number of sampling points for each gesture frame) and bit cardinality (*i.e.,* the precision of the measurements) on

the execution speed and accuracy of 3D Nearest Neighbors classifiers. They observe that, past a certain threshold, higher dimensionality and bit cardinality do not lead to significantly higher recognition rates. By carefully selecting the values for gesture dimensionality and bit cardinality, it is thus possible to significantly reduce execution speed while still reaching (almost) peak accuracy. They also note that accuracy generally increases with the number of training samples, but decreases with the size of the gesture set.

2.4 LMC Gesture Sets

From the Targeted Literature Review conducted in Section 2.3, a series of LMC-based gesture sets has been identified, which enables us to perform a comparative analysis of gestures that are typically, frequently manipulated by LMC-based UIs. Among them, we highlight two significant ones that will be used later on.

For example, the **SHREC2019** dataset [17] served as a benchmarking for the Eurographics 2019 SHape Retrieval Contest (SHREC) track on online gesture recognition to detect command gestures from hands' movements in a virtual reality context. The proposed dataset is topical due to its recency. The emergence of this periodic contest in the field of computer graphics results in the indexation of this dataset as a reference dataset. This dataset consists of 195 3D trajectories performed by 13 participants with the whole hand. Each trajectory contains a sequence of 3D points and quaternions designating one of five iconic gestures: "Cross" (X), "Circle" (O), "V-mark" (V), "Caret" (/\), and "Square" ([])).

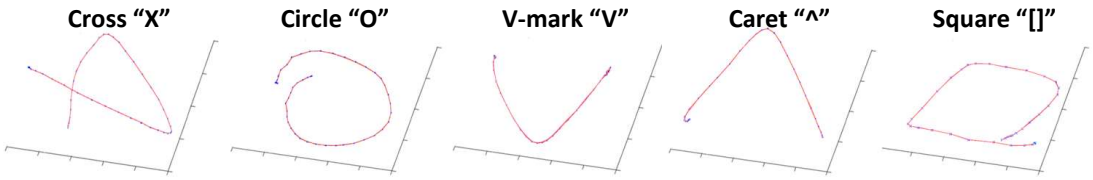


Figure 2.3: The **SHREC2019** dataset [17].

The **3DTCGS** dataset [18] is employed to test the classification performance of the *3 cent* [18] recognizer on interface command gestures. Data were provided in a segmented form which makes them easier to exploit during experimentation for the classification task. This dataset contains gestures of varying complexity (Figure 2.3), ranging from the simplest (*e.g.,* a "3D swipe") to the most complex one (*e.g.,* a "3D spiral" is challenging because it can be drawn in a co-planar way or with varying depth). Some other gestures follow a similar trajectory, but with a different direction (*e.g.,* "left-swipe" *vs.* "right-swipe", "arc3Dleft" *vs.* "arc3Dright"), which is considered for testing the direction-invariance property. Each subject produces one sample per gesture which makes the user-independent evaluation particularly challenging. Authors asked participants to issue short iconic gestures with the dominant hand forefinger. The gathered data consist of 347 sequences of 3D coordinates. These data are combined with a timestamp, describing 26 gesture classes performed by 14 subjects and recorded with a Leap Motion. They are clustered in five categories (Figure 2.4): semi-circular arcs (*i.e.,* "arc3Dleft", "arc3Dright"), symbols drawn in a 3D space (*i.e.,* "caret", "check", "curly-bracket-left", "curly-bracket-right", "delete", "pigtail", "square-bracket-left", "square-bracket-right", "star", "v", "x"), Simple geometric figures (*i.e.,* "left-swipe", "right-swipe", "circle", "rectangle", "zig-zag"), 3D polygonal chains (*i.e.,* "poly3Dxyz", "poly3Dxzy", "poly3Dyxz", "poly3Dyzx", "poly3Dzxy", "poly3Dzyx"), and a 3D spiral.

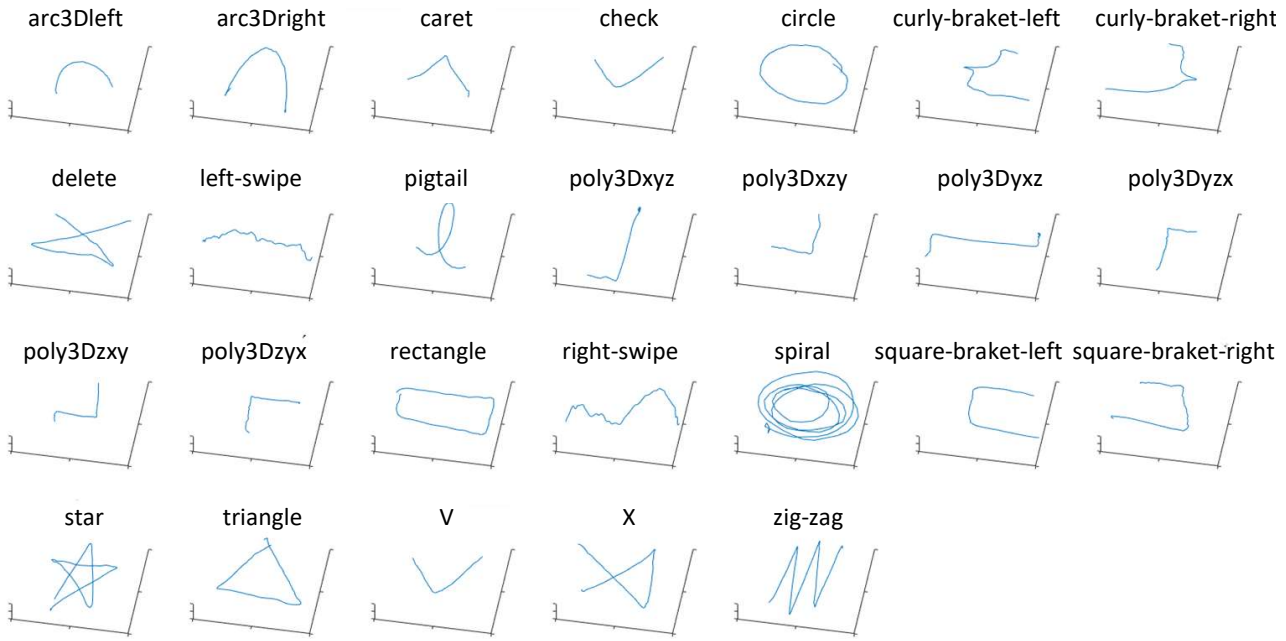


Figure 2.4: The **3DTCGS** dataset [18].

2.5 Summary

Apart from the recognizers adopting a template matching, such as the recognizers belonging to the $-family, most of them are too complex for incorporating them into a generic framework. Indeed, besides their intrinsic computational complexity, most of them require intensive training before they can be used to recognize a given set of gestures. Furthermore, adding a new gesture to an existing gesture set requires re-training the recognizer. Being able to add custom gestures without having to re-train the recognizer is a desirable property, as the literature shows that recognizers' accuracy is better with user-dependent gestures [118]. We retain template-based approaches (Section 2.2) because, contrarily to others, they provide the means for satisfying the ISO/IEC properties selected in Section 1.1:

- *Modifiability*: such recognizers can easily accommodate any change in the gesture set: adding a new template or a gesture class, removing a template or a class, or modifying them do not alter the algorithm and do not require complete re-modeling, re-training or pre-processing, apart those concerned by the local modifications.
- *Reusability*: such recognizers can be reused straightforwardly in various configurations of LMC-based gestures without modifying their code.
- *Replaceability*: such recognizers are interchangeable as they pretty much work on the same gesture representation, apart from their format. Thus, replacing a recognizer by another one does not affect the entire framework or recognition.
- *Adaptability*: such recognizers are directly adapted to the current gesture set without needing to perform calibration, tweaking, or parametrization to adapt to a new context.
- *Modularity*: thanks to the incremental development that introduced variations of recognizers based on previous versions, each recognizer can be implemented into a single module and variations can be created on top of them without affecting the overall architecture.

Chapter 3

Framework Development

Chapter 2 identified a series of recognition techniques targeting the LMC. Most of them are built and tested for one specific application and/or for a single context of use, making it hard for developers to (re-)use these techniques in another application or context of use. In order to address the ISO quality properties defined in Section 1.1, this third chapter describes our framework[1] for engineering LMC-based gesture UIs as follows: Section 3.1 defines its requirements, Section 3.2 describes its software architecture, and Section 3.3 illustrates how the framework can be used to develop a small application.

3.1 Requirements

To leverage the shortcomings of work related in LMC-based recognition, the framework should satisfy the ISO/IEC 205010 [50] quality properties selected in Section 1.1 as follows:

1. **Compatibility**
   - *Co-existence*: the framework should be able to work properly alongside other applications that are also using the LMC.
   - *Interoperability*: aside from supporting a variety of gesture sets, the framework should use a standardized format for skeleton-based gesture sets.
2. **Maintainability**
   - *Modularity*: the framework should be composed of independent modules with clear, separate roles.
   - *Reusability*: the framework should allow front-end developers to add gesture support to any application easily.
   - *Modifiability*: the framework should allow developers to add new, more powerful modules (*e.g.,* recognizers) to the framework without modifying their application or the other modules.
   - *Testability*: the framework should provide a reliable way to compare the accuracy of gesture recognizers.
3. **Portability**
   - *Adaptability*: the framework should be quickly configurable to the needs of a specific user, application, and system.
   - *Installability*: the framework should be multi-platform and relatively easy to install.
   - *Replaceability*: developers should be able to replace any module in the framework, such as changing a recognizer or a gesture set.

[1] The framework was developed in collaboration with Bastien Nothomb [112]. Appendix A details the distribution of the work between the two of us.

3.2 Software Architecture

We designed the framework as an intermediate layer between an application and a sensor. Isolating the UI from the recognizer and the sensor makes it easy for a developer to write a new application without focusing on the recognition techniques. In the same way, switching the sensor with another one (*e.g.,* going from an LMC to a Kinect device), or using a different recognizer should not force developers to modify their application.

We chose JavaScript as the programming language for four main reasons: (1) it is an expressive, powerful, and cross-platform language that enables applications to run directly in a web browser without requiring any setup; (2) it is used in many domains of applications, ranging from games to virtual reality, thus opening the range of potential applications; (3) some gesture recognizers that are candidates for incorporation and extension in our framework are available in this language; and (4) it is a popular language, with a lot of highly maintained libraries.

Following Pree's notation [92], the framework can be divided into two parts: frozen and hot spots. Frozen spots are its "fixed" parts. They are the same for all of its instances. On the other hand, hot spots are generic parts that can be easily modified to better suit the context of use. Because it both provides a selection of predefined components (black box) and allows developers to write new components that better suit their needs (white box), the framework can be considered as gray-box. In the remainder of this section, we describe the overall software architecture of the framework: its frozen and hot spots, as well as how it can be configured to suit the needs of a specific context of use [16].

3.2.1 Overview

Figure 3.1 shows the overall architecture of the framework. Messages sent across the modules are the following:

1. Raw data from the sensor.
2. A standardized frame from the sensor (see Section 3.2.2).
3. A Sample object (see Section 3.2.2).
4. A JavaScript object with three properties: (1) the type of gesture, *i.e.,* static or dynamic; (2) the name of the gesture; and (3), data from the analyzer if the gesture is static.
5. A WebSocket message of type data, containing frames, static and dynamic gestures.
6. A WebSocket message of type operation (see Section 3.2.2).

When initialized, the frame-processor loads one gesture set to train the classifier and another for the recognizer. Depending on its configuration, it can train them directly with all of the gestures or only when requested by the application.

Each time the sensor-interface produces a frame, it does two things: (1) transmit raw sensor data to the WebSocket server, which is then forwarded to the app-interface; and (2) send a standardized frame to the frame-processor which then attempts to classify the pose (*i.e.,* static gesture) performed by the user. Depending on the type of pose, two things can happen: (1) the frame is forwarded to the analyzer if the application previously informed the framework that it wants to handle the pose; or (2) the frame is sent to the segmenter. In the first case, the pose and extracted data are sent to the application. In the second one, if the segmenter determines that the user has performed a gesture, it sends the gestural data to the recognizer. If it recognizes a gesture, it sends its name to the application.

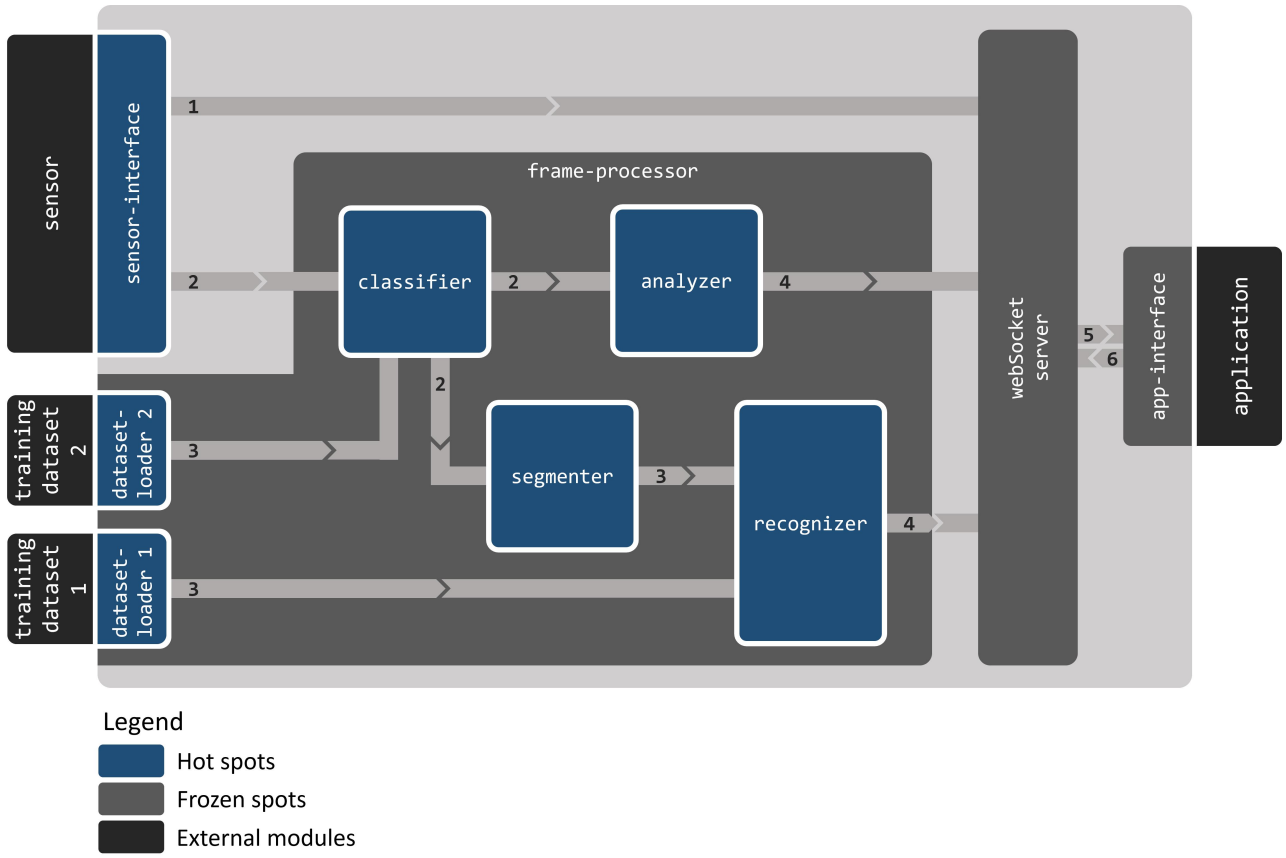


Figure 3.1: Software architecture of the framework.

3.2.2 Frozen Spots

Some parts of the framework are fixed and standardized to maximize reusability across multiple projects, at the cost of lower flexibility. Their source code is available in Appendix F.1.1.

Frame

The frame.js module represents a standardized frame from the sensor. One frame contains skeleton data extracted from the sensor at one particular point in time. All frames have the following format:

```
timestamp: integer - milliseconds since the Epoch
hasLeftHand: boolean
hasRightHand: boolean
articulations: array of Articulation objects
    label: string - e.g., "rightPalmPosition"
    points: array of Point objects
        x: float
        y: float
        z: float
        t: integer - milliseconds since the Epoch
```

The key-value system allows developers to easily add or remove articulations, depending on the context of use (*e.g.,* the LMC only provides forearm articulations, while other sensors, such as the Kinect, could provide articulations of the whole body).

Gestures and Gesture Sets

Figure 3.2 depicts the UML V2.5 [15] Class diagram of the modular data structure used in the framework. Any dataset can be used in the framework. They just need to be converted into GestureSet objects. A GestureSet object can contain samples (also known as templates) of two types: (1) PoseData, for static gestures; and (2) StrokeData, for dynamic gestures. PoseData objects consist of a single frame that represents a hand pose. On the other hand, StrokeData objects can comprise of multiple paths, where a path corresponds to the trajectory of a specific articulation (*e.g.,* the tip of the right index). Paths are in turn composed of a series of strokes.

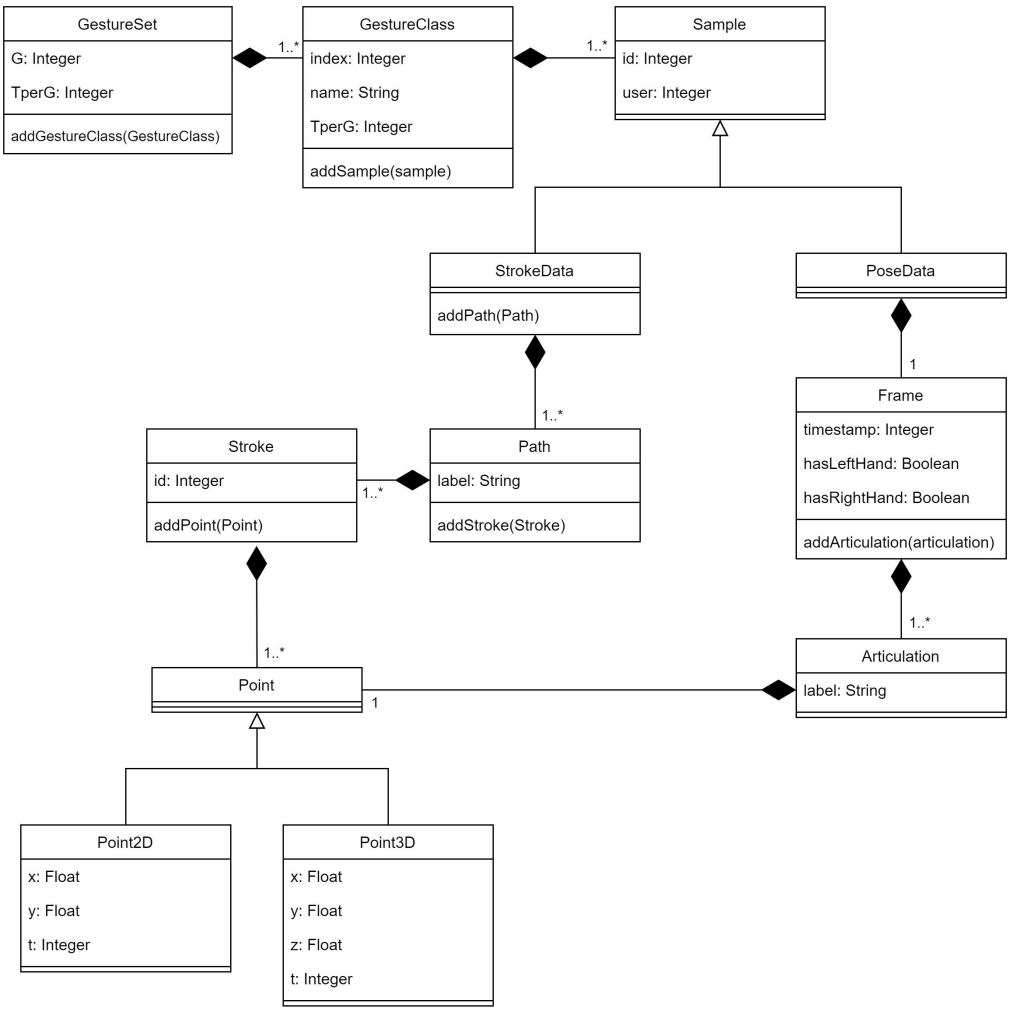


Figure 3.2: UML V2.5 Class Diagram of the Framework.

WebSocket Server

index.js is the main module of the framework. It loads the framework configuration and initializes the sensor module (see 3.2.3). It also sets up a WebSocket server, used to both receive and send messages to the client application. In the current version of the framework, clients only send operation messages to the server. There are four types of operations: addGesture, addPose, removeGesture and removePose. These messages allow the client application to control which gestures and/or poses should be recognized by the framework, which improves recognition accuracy. The server only sends data messages to the client application. Their payload can be of three types: frame, gesture, or pose.

Frame Processor

The frame-processor.js module is at the core of the framework, as it coordinates most of the modules to extract data from sensor frames and determine whether the user is performing a pose or a gesture. For more information, see Section 3.2.1 and Figure 3.1.

Application Interface

The app-interface.js module is used as an interface between the application and the recognition framework. It contains the GestureHandler class, which has the following methods:

- onFrame(callback): execute the callback each time a frame is received from the sensor. This can be used to display the position of the fingers on the screen in real-time.
- onGesture(gesture, callback): execute the callback each time the (dynamic) gesture is detected. This can be used by an application to associate an action with a specific gesture.
- onEachGesture(callback): execute the callback each time any gesture is detected. This can be used to perform an action regardless of the gesture performed.
- removeGestureHandler(gesture): remove the callback associated to the gesture.
- onPose(pose, callback): execute the callback each time the pose (*i.e.,* static gesture) is detected. This can be used to execute a "continuous" action while the pose is detected (*e.g.,* rotate an image depending on the rotation of the hand).
- removePoseHandler(pose): remove the callback associated to the pose.
- disconnect(callback): disconnect from the gesture recognizer.

Once instantiated, a GestureHandler object regularly attempts to connect to the gesture recognition server using WebSocket. Using WebSocket allows the server to directly push data, such as frames, gestures, or poses, to the client. Because the client does not need to regularly poll data from the server, there is less delay between an action and its effect on the UI.

3.2.3 Hot Spots

The framework is composed of six different types of interchangeable modules. Any instance of the framework must be composed of one module of each type, but "placeholder" modules are provided for situations where a module is not necessary. This section describes, for each type of module, their role, and the methods that they have to implement. The abstract classes are available in Appendix F.1.2, while the source code of the modules that we have implemented can be found in Appendix F.2 to F.7.

Sensors

Sensor modules serve as an interface between the framework and a physical sensor (*e.g.,* an LMC or a Kinect device). Every sensor has to extend the AbstractSensor class and implement its two methods:

- loop(callback): this method transforms frames from the sensor into a standardized format. Each time it receives a frame from the sensor, it executes the callback with the standardized frame as an argument.
- stop(): stop processing frames from the sensor.

For the moment, we provide a module for the LMC that calls the Leap Motion API to fetch frames at regular intervals (their duration is adjustable in the configuration file).

Datasets

Dataset modules transform a dataset into a standardized format compatible with the framework (see Figure 3.2). They only need to contain one method:

- loadDataset(name, directory): this method takes the name of the dataset and the path to the directory where it is stored. It returns a GestureSet object representing the dataset.

Classifiers

Classifier modules determine the hand pose of the user. They have to extend the AbstractClassifier class, which contains three methods:

- addPose(name, frame): add a frame corresponding to a pose to the training set of the classifier.
- removePose(name): remove a pose from the training set of the classifier.
- classify(frame): determine the name of the pose that corresponds to the frame given as argument.

We have implemented two 3D pose classifiers:

- P^3+. A 3D version of Vatavu's *$P+* algorithm [119]. It is position, direction, and scale-invariant, but not rotation invariant. It is described in more details later in this section.
- GPSDa. Proposed by Prof. Paolo Roselli[2], GPSDa is a position, scale, direction and rotation invariant recognizer. Static gestures, also called hand poses, are represented as *Point a-Shapes*. The Point a-Shape $PS_{P,a}$ of a static gesture P is the list of the normalized reciprocal a-distances between each pair of points, arranged in non-decreasing order. If
  - $P = (P_1, ..., P_N)$, where $P_i = (x, y, z)$, is the list of 3D points of gesture P;
  - $|P_i - P_j| = \sqrt{(P_i.x - P_j.x)^2 + (P_i.y - P_j.y)^2 + (P_i.z - P_j.z)^2}$ is the Euclidean distance between points P_i and P_j;
  - $B_P = \frac{1}{N} \sum_{i=1}^{N} P_i$ is the barycentre of the points of P;
  - $MD_{P,B,a} = \frac{1}{N} \sum_{i=1}^{N} |P_i - B_P|^a$ is the mean a-distance between the points P_i and their barycentre B_P.

  Then, the Point a-Shape of gesture P is defined as:

$$PS_{P,a} = Sort\left(\left(\frac{|P_i - P_j|^a}{MD_{P,B,a}}\right)_{1 \le i < j \le N}\right) = (\sigma_{P_1}, ...\sigma_{P_{\frac{N(N-1)}{2}}})$$

  The Global Point Shape a-Distance (GPSD) between a gesture G and a template T can now be defined as:

$$GPSD_{G,T,a} = \sum_{h=1}^{\frac{N(N-1)}{2}} |\sigma_{G_h} - \sigma_{T_h}|$$

  Note that a is a parameter that impacts recognition sensibility. Gesture recognition is a two steps process. The recognizer first computes the Point a-Shape of the gesture. Then, for each template, it calculates the GPSD between the template and the gesture and returns the template which minimizes it.

[2]This method is proposed by Prof. Paolo Roselli, Lecturer at Université catholique de Louvain and Senior Research Associate, Università di Rome "Tor Vergata", and implemented for the first time in the context of this framework.

Analyzers

The role of an analyzer module is to extract data from a frame after a pose has been recognized. This data is then sent to the application, along with the pose that was recognized. Every analyzer has to implement the two methods of the AbstractAnalyzer class:

- analyze(frame): extract and return relevant information from a frame.
- reset(): reset the internal state of the analyzer.

We provide one module of this type, basic-analyzer. It extracts four values from each frame: (1) rotation, the rotation of the hand compared to the previous frame; (2) pinch, the ratio of the distances between the index and the thumb of the current and the previous frames; (3) translation, the displacement of the palm compared to the previous frame; and (4) thumbVector, a 3D vector that gives the orientation of the thumb.

Segmenters

The role of segmenters is to determine if a set of frames consists of a gesture. If the set corresponds to a gesture, it sent directly to the recognizer. Each segmenter must extend the AbstractSegmenter class and implement its two methods:

- segment(frame): this method processes a frame and returns a StrokeData object if the frame marks the end of a gesture.
- notifyRecognition(): notify the segmenter that a gesture has been recognized. The segmenter can use this information to avoid unintended gestures that could occur due to coarticulation between two subsequent gestures.

We designed 3 different segmenters that each have their pros and cons. Please note that this list is non-exhaustive, as gesture segmentation [62] is a deep and interesting problem which could be the subject of an entire thesis.

- Left-hand segmenter. This segmenter uses the presence of the left hand in the sensor's field of view to trigger the buffering of frames. As soon as the user removes his left hand, it sends the collected gesture data to the recognizer. This method is easy to implement and usually provides accurate segmentation, as users can directly tell the framework when they are performing a gesture. However, it prevents them to use their left hand for other tasks and makes accurate recognition of two-handed gestures more difficult.
- Zoning segmenter. This approach is conceptually very similar to the left-hand approach. Frames from the sensor are buffered while at least one hand is within a 3D zone in space, delimited by a set of boundaries. If both hands leave the zone, or if the buffer is full, the gesture data is sent to the recognizer. In this approach, gestures can be performed with either one hand or both hands, without requiring both of them to be free at all times. However, the size of the bounding box limits the range of movement. If it is too small, users might not be able to perform gestures accurately. On the other hand, if it is too large, unintended gestures might be picked up by the segmenter.
- Window segmenter. This segmenter attempts to overcome the limitations of both the zoning and the left-hand segmenters. It uses a fixed-size buffer called the window. Each time a new frame is received, it is pushed in the window, while the oldest frame is removed. Every fixed number of frames, the buffered gesture data is sent to the recognizer. Because it does not impose a limit on the range of motion and does not require both hands to be available at all times, this approach might seem like the best of both worlds. However, it

still suffers from a few issues, some of which can be alleviated at some cost. First, its fixed window size limits the maximum duration of the gestures. Second, because it regularly sends sets of frames to the recognizer, regardless of whether they represent an intended gesture, it causes more CPU strain and may cause the detection of unintended gestures. We introduce two changes to fix this issue: (1) the segmenter waits for some time before forwarding gestural data to the recognizer after a gesture is detected; and (2) it does not forward the data if the amplitude of the movement is below a certain threshold.

Recognizers

Recognizer modules are very similar to the classifiers. Their role is to determine the name of the gesture performed by the user. Recognizers should implement the three methods of AbstractRecognizer:

- addGesture(name, sample): add a gesture to the training set of this recognizer.
- removeGesture(name): remove a gesture from the training set of this recognizer.
- classify(sample): determine the name of the gesture that corresponds to the sample given as argument.

We have implemented six 3D gesture recognizers:

- 3 cent. An optimized 3D port of Wobbrock *et al.*'s *$1* [133] by Caputo *et al.* [18]. As opposed to the other recognizers presented in this section, it only supports single-stroke gestures. To recognize a gesture, it first applies a three-step pre-processing: (1) resample the gesture to a fixed number of equidistant points; (2) scale the gesture uniformly to a standard unit length; (3) translate the centroid of the point cloud to the origin, *i.e.,* $(0, 0, 0)$. Then, it computes the distance between this gesture and each template, as the sum of the squared distances of corresponding points. It returns the template with the shortest distance. This recognizer is position and scale-invariant, but neither rotation nor direction invariant.
- P^3. An implementation of Vatavu *et al.*'s *$P* [121] that supports 3D gestures. To support multi-stroke gestures while keeping memory usage and execution time low, gestures are represented as unordered sets of points, called *point clouds*. Gesture recognition happens in two phases: normalization and cloud-matching. The normalization process is similar to other $-family algorithms and is divided into three steps: (1) resample the gesture to a fixed number of equidistant points; (2) scale the gesture uniformly to keep its shape; (3) translate the centroid of the point cloud to the origin, *i.e.,* $(0, 0, 0)$. The cloud-matching phase matches a gesture's point cloud to the point cloud of each template by associating each point from the template to exactly one point from the gesture. It then computes the resulting distance, as the sum of Euclidean distances between all pairs of matching points, and returns the template that minimizes it. This recognizer is position, direction, and scale-invariant, but not rotation invariant.
- P^3+. A more accurate version of *P^3*, adapted from Vatavu's *$P+* [119]. It brings two key improvements to *$P*: (1) each point from the template can now be matched to more than one point of the gesture; (2) the distance between a gesture and a template takes into account the connections between consecutive points (in the form of their turning angles). This recognizer has the same invariance properties as *P^3*.
- P^3+X. A variation of *P^3+*, with partial direction invariance. In the context of large UIs, gesture direction may be key to telling two gestures apart (*e.g.,* swipe left vs. swipe right). However, true direction invariance may reduce accuracy for gestures that may be

performed in any direction, such as letters or symbols. To maximize accuracy, *$P^3+* keeps track of conflicting templates, *i.e.,* templates that represent the same gesture but drawn in different directions. Gesture recognition is similar to *$P^3+*. However, depending on the template that is returned, two scenarios can happen: (1) the template is not marked as "conflicting", it is thus returned directly; (2) the template is marked as "conflicting", the direction of the gesture is compared with each conflicting template and the closest one is returned. This recognizer position and scale-invariant, partially direction invariant, but not rotation invariant.

- Q^3. A 3D version of Vatavu *et al.*'s *$Q* [122], which achieved a 46X speedup on average over *$P* with no loss of accuracy. It brings two major changes to *P^3*: (1) early abandonment of templates, as soon as the distance becomes greater than the current shortest distance; (2) each point cloud has a 16×16×16 3D look-up table (LUT), where each location (x, y, z) contains the index of its closest point. Using look-up tables, a lower bound of the distance between a gesture and a template can be computed with a time complexity of $O(n)$. If this lower bound is greater than the current shortest distance, it does not need to compute the exact distance. This recognizer has the same invariance properties as *P^3*.
- Jackknife. A general-purpose 3D gesture recognizer [111] that supports any number of articulations. Unlike most of the dollar-family recognizers, it represents gestures as time-series. It uses the nearest neighbor approach, where it compares a gesture to each template, and returns the closest one. Unlike the recognizers described before, Jackknife uses *dynamic time warping* to measure the distance between a template and a gesture. However, it applies a series of correction factors to take into account differences in gesture scale and span. As a result, this recognizer is both position and scale-invariant, but neither direction nor rotation invariant.

3.2.4 Configuration File

The config.js file is used to configure different aspects of the framework so that it suits the needs of an application. For most hot spots, developers can simply select which module they want to use and set its options. The following example shows how one can configure a recognizer, but it works similarly for sensors, classifiers, analyzers and segmenters. A configuration file is shown in Appendix F.1.3.

```
config.recognizer = {
    module: P3DollarPlusXRecognizer,
    options: {
        palmThreshold: 50,
        fingerThreshold: 15,
        samplingPoints: 16,
        articulations: palms.right,
    }
}
```

Configuring datasets is slightly different. There is no options field but the framework offers three parameters: useCustomTemplatesPerClass and templatesPerClass control the number of templates that are used to train the recognizers, whereas aggregateClasses is used to combine and rename gesture classes, as well as select only a subset of the gesture set.

```
config.datasets.gesture = {
    directory: __dirname + "/datasets/gesture",
    loader: LeapMotionDatasetLoader,
    name: "guinevere",
    useCustomTemplatesPerClass: true,
    templatesPerClass: 16,
    aggregateClasses: [
        { name: "rhand_uswipe", classes: ["swipe_index_up", "swipe_palm_up"] },
        { name: "rhand_dswipe", classes: ["swipe_index_down"] },
        { name: "rhand_lswipe", classes: ["swipe_index_left"] },
        { name: "rhand_rswipe", classes: ["swipe_index_right"] },
        { name: "rindex_airtap", classes: ["tap"] },
    ]
}
```

3.3 Use-Case: the DICOM Viewer

3.3.1 Overview

During an operation, surgeons often have to look at medical images of the patient. However, keeping computer controls sterilized is difficult and it is not always possible to have another person dedicated to the manipulation of images. Bastien Nothomb's DICOM viewer [112] aims at solving these issues by bringing gestural interaction into the operating room. It uses our framework to allow surgeons to view and manipulate DICOM images without any contact (see Figure 3.3).

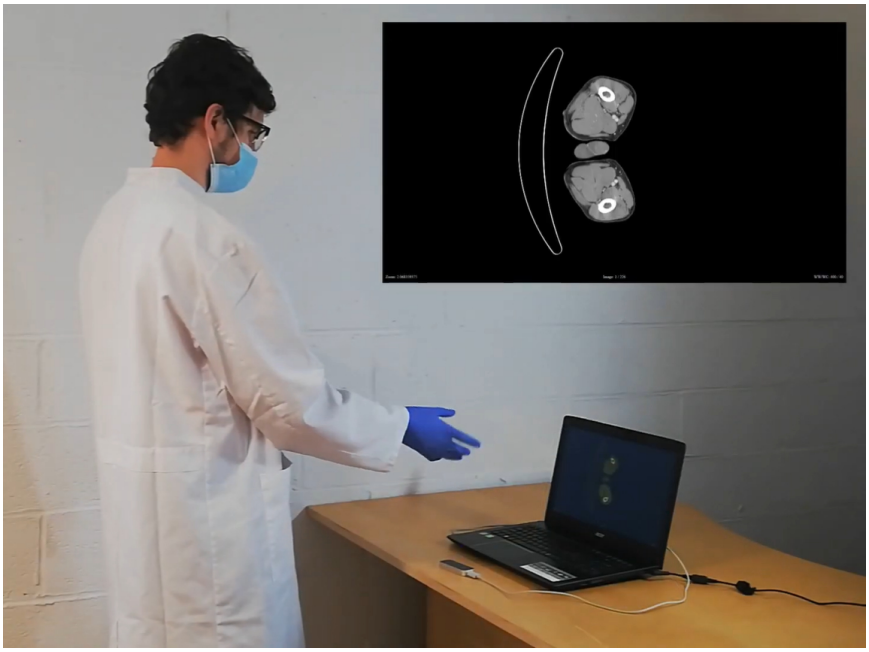


Figure 3.3: The DICOM application.

The application is still a prototype but it already offers a wide range of functionalities. In its current state, it shows a list of DICOM "studies" corresponding to one patient. Each study corresponds to an examination of the patient, which can contain one or more images. Surgeons

can navigate from one study to the next using "swipe" gestures. Within a study, multiple actions are possible, such as manipulating the image or moving to the previous/next one. See Table 3.1 for the complete list of supported actions.

3.3.2 Gesture Set

Table 3.1 shows the gestures associated with each of the 14 actions supported by the current version of the DICOM viewer. The 14 actions and their corresponding gestures were selected based on a gesture elicitation study by Madapana *et al.* [76].

Category	Action	Gesture
Studies navigation	Next study	Swipe left
	Previous study	Swipe right
Image animation	Forward	Swipe up
	Backward	Swipe down
Image manipulation	90° rotation clockwise	Turn invisible button clockwise
	90° rotation anti-clockwise	Turn invisible button anti-clockwise
	Zoom in	Pinch out
	Zoom out	Pinch in
	Increase contrast	C shape w/ the left hand, swipe up w/ the right hand
	Decrease contrast	C shape w/ the left hand, swipe down w/ the right hand
	Pan up	Open hand in up direction
	Pan down	Open hand in down direction
	Pan left	Open hand in left direction
	Pan right	Open hand in right direction

Table 3.1: Gesture set of the DICOM application.

3.3.3 Configuration and Efficacy of the Framework

Figure 3.4 shows the configuration of the framework for the DICOM viewer. The classifier and analyzer modules are not needed because all of the 14 supported gestures are dynamic. It is configured with the Jackknife recognizer, as it reaches high accuracy, even on a large number of gesture classes [111].

The number of lines of code (LOC) of the application was greatly reduced by the use of the framework. In its current state, the DICOM viewer fits into 375 LOC. Without the framework, this number would possibly double to about 770 LOC. Indeed, by taking a look at the source code of the framework, we can observe that, together, the modules that are used in this instantiation are comprised of 397 LOC, distributed into:

- 106 LOC to connect to the LMC and extract data in the correct format;
- 115 LOC to load the gesture dataset;
- 82 LOC for the zoning segmenter;
- 94 LOC for the Jackknife recognizer.

Overall, using the framework has enabled both substantial time-savings and a significant reduction in the complexity of the application.

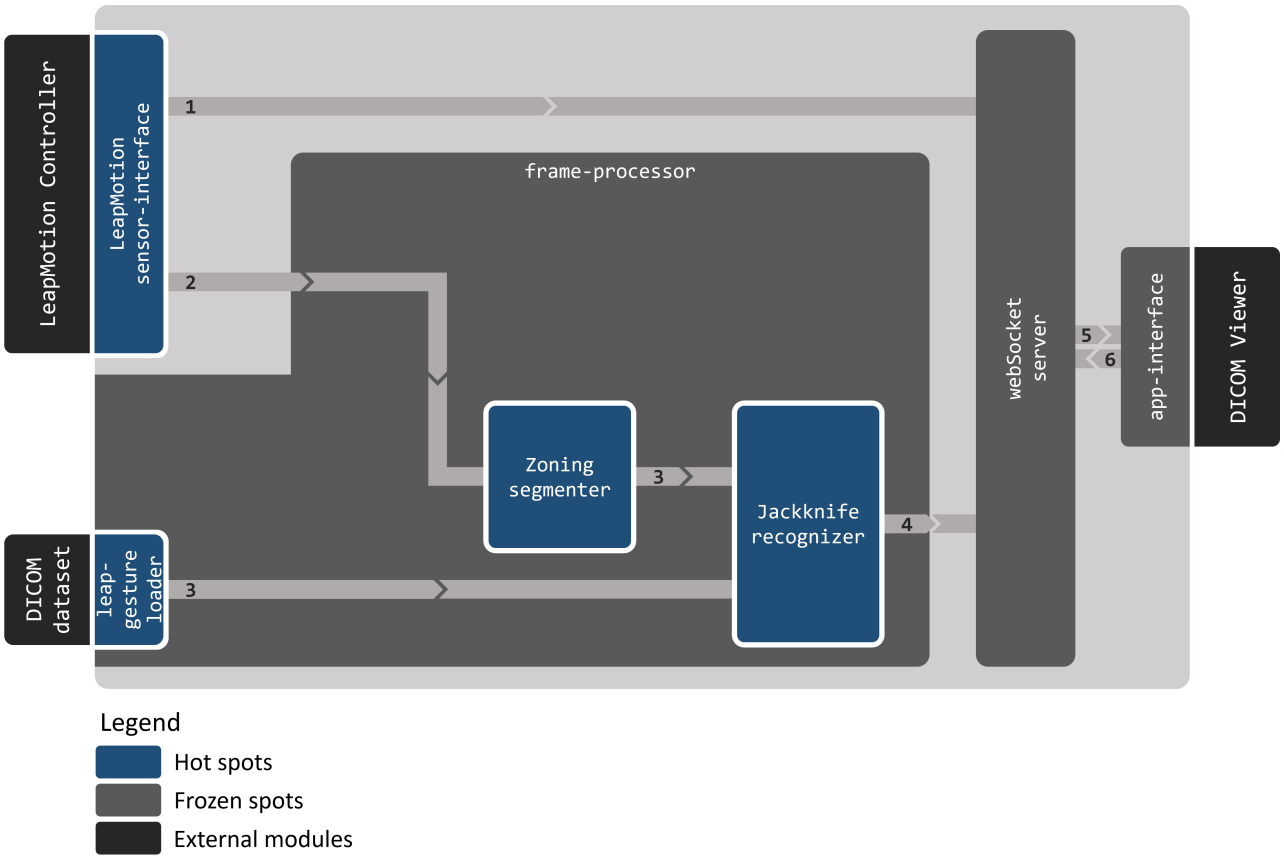


Figure 3.4: Configuration of the framework for the DICOM application.

3.4 Conclusion

This chapter presented the complete development process of the framework for gesture recognition. It first laid out a series of requirements to be satisfied by the framework, based on three ISO/IEC 205010 [50] quality properties. It then detailed its architecture: the gesture recognition pipeline and the modules that were implemented. All of this led up to a first instantiation of the framework, in the form of a small-scale application: the DICOM viewer. Next, chapters 4 and 5 will showcase two larger-scale applications of the framework.

Chapter 4

Benchmarking of Recognizers

The framework for engineering LMC-based UIs introduced in Chapter 2 was illustrated on a first, small-scale application, the DICOM viewer (Section 3.3). This chapter presents a second application of this framework that is different from developing a gesture-based UI: a benchmarking of gesture recognizers. Section 4.1 describes the configuration of the framework, the recognizers, and the evaluation procedure used for this purpose. The results are then summarized and discussed in Section 4.2.

4.1 Benchmark

4.1.1 Benchmarking Tool

The software architecture of the framework (see Figure 3.1) allows configuring modules for a variety of purposes, such as, obviously, developing a gesture-based UI for an interactive application with an LMC, but also for other purposes, such as testing a particular recognizer against one or many gesture sets, testing a particular gesture set across recognizers, and designing gestures for the LMC by managing gesture sets. In particular, benchmarking recognizers is aimed at comparing the overall behavior of several recognizers on several gesture sets to identify which one is the most accurate in terms of recognition rate and the most efficient in terms of execution time. For this purpose, a benchmarking module is integrated into the framework to compare both static and dynamic gesture recognizers with limited effort (Figure 4.1). Similarly to the gesture recognition part, it holds a configuration file that allows testers to select and configure a series of recognizers and gesture sets. As this chapter focuses exclusively on dynamic gesture recognizers, only the bottom part of the pipeline shown in Figure 4.1 is used.

4.1.2 Recognizers

In addition to the *$P*^{3}, *$P*^{3}+, and *$Q*^{3} recognizers implemented and described in Section 3.2.3, we also incorporated in the framework four recognizers developed by Nathan Magrofuoco and Mehdi Ousmer, thus demonstrating adaptability, a sub-factor of comptatibility.

Rubine3D

Inspired by the iGesture framework [105], Rubine3D is a feature-based recognizer that combines a set of three individual 2D Rubine recognizers [97], one for each plane XY, YZ, and ZX. 3D trajectories are projected onto the three planes by transforming each of their points to the

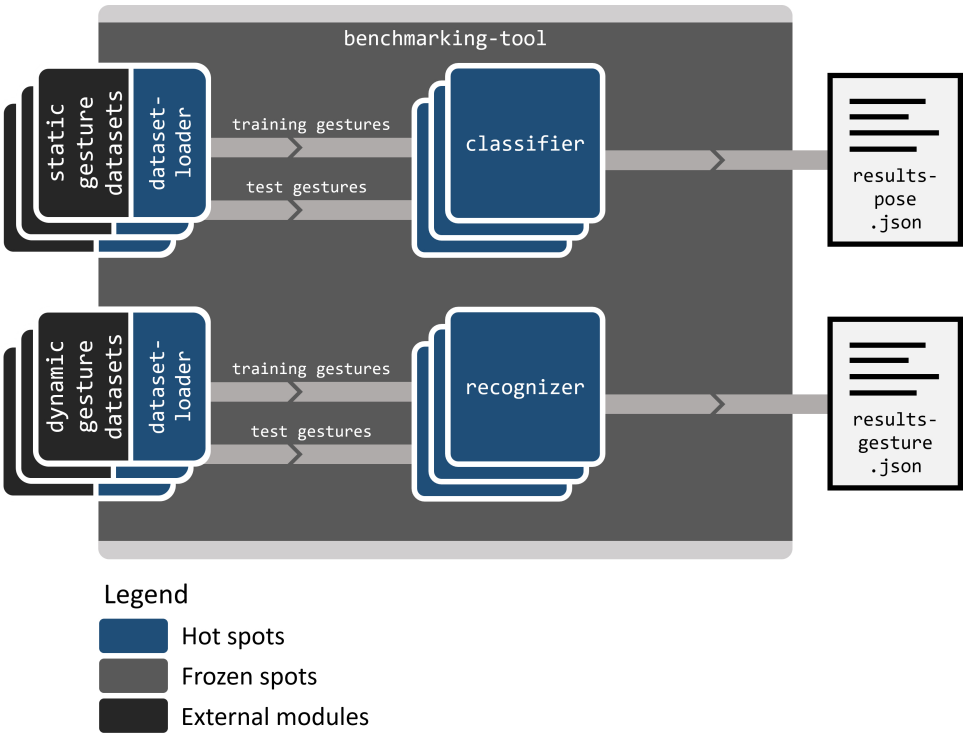


Figure 4.1: Configuration of the framework for the benchmarking of recognizers.

three coordinate system planes (XY, YZ, ZX). If $M(i, j, h)$ is a point of the 3D trajectory, then $P(i, j)$, $Q(j, h)$, and $R(h, i)$ are the 2D point coordinates of the projection onto (XY, YZ, ZX).

Before it can recognize 3D trajectories, Rubine3D has to pre-process the training templates. Each template is first projected onto the three planes. Then, for each plane, the recognizer compute weights for the features of each gesture class in five steps: (1) the raw data is pre-processed by scaling and filtering its points; (2) the 13 original features ($f_1, .., f_{13}$) defined by Rubine [97] are computed for each template; (3) from these features we compute the mean feature vector and the covariance matrix of each gesture class; (4) we compute the common covariance matrix and its inverse matrix; and (5) we estimate the weights for each gesture class from the inverse matrix. Once Rubine3D is trained, there are several steps to determine the possible class of a candidate gesture. The gesture is first projected onto each plane and three feature vectors are extracted from the three projections. For each projection, we return the gesture class that maximizes the similarity score (computed by combining the weights of the gesture class with the feature vector of the gesture). The results from the three projections are then merged using a heuristic inspired by iGesture. If the three projections return the same gesture class, it is designated as the final result. Otherwise, if there are two or three different classes, we compute the similarity score of these classes for the three planes. The scores of each plane are weighted (W_{XY}=0.4, W_{YZ}=0.3, W_{ZX}=0.3) to factor in the relative difficulty of producing gestures in that specific plane [114]. We return the gesture class which maximizes the sum of the three similarity scores.

Rubine-Sheng

Rubine-Sheng (RS) is a 3D recognizer inspired by Rubine. It supports 3D gestures by adding three new features proposed in the AdaBoost recognizer [104] to Rubine's existing 13 features. Aside from this difference, it is extremely similar to its 2D counterpart for both the pre-processing of the templates and gesture recognition.

$F

The *$F* recognizer adds *$P+*'s flexible cloud matching [119] to *$P³*. As for most $-recognizers, the points of the candidate and the templates are resampled to equidistantly-spaced points, scaled within a unit box, and translated so their centroid is at the origin $(0, 0, 0)$. The template with the lowest dissimilarity score is considered as the best matching template for the candidate. As opposed to *$P³*, *$F*'s cloud matching is flexible, as points can be matched to more than one point. It consists of matching the points from the first cloud with their closest point from the second cloud, then matching the points from the second cloud that have not been matched yet with their closest point in the first cloud.

FreeHandUni

The FreeHandUni (*FH*) recognizer is derived from Vatavu *et al.*'s Free-Hand recognizer [32], which extends the *$P* 2D gesture recognizer [121] to support 3D hand gestures. FreeHandUni is adapted to meet our need for recognizing the unipath character of 3D trajectories by replacing the hand pose structure with a 3D point structure (x, y, z). With this modification, FreehandUni corresponds to an improvement of *$P³*, using a flexible cloud matching based on a one-to-many alignment between points [119]. The pre-processing stays the same but the matching process is more flexible: each point of the template cloud is matched with the closest point from the candidate cloud, then the remaining points from the candidate cloud are matched with the closest point from the template cloud. It returns the gesture class with the lowest dissimilarity score. FreeHandUni is different from *$F* in that the early abandoning is not implemented, to align the computational complexity to *$P³*.

4.1.3 Experiment

We evaluated the seven resulting recognizers, the three that we implemented, and the four that we integrated via their configuration files, in the traditional user-independent procedure [5,6,119,121].

Design

Our study was within-subjects with two independent variables:

1. RECOGNIZER: nominal variable with 7 conditions, representing the various recognizers implemented for recognizing 3D trajectories: *$P³*, *$P³+*, *$Q³*, *$F*, *FH* (FreeHandUni), *R3D* (Rubine3D), and *RS* (Rubine-Sheng).
2. DATASET: nominal variable with 2 conditions, representing two datasets *i.e.,* SHREC2019 [17] and 3DTCGS [18]. Both datasets are described in Section 2.4.

Procedures and Quantitative Measures

We compute the *recognition rate* (the ratio of positive recognitions divided by the total number of trials) and the *execution time* (the time for pre-processing a candidate gesture and determining its result class in milliseconds) for the 7 (RECOGNIZER) × 2 (DATASET) = 14 basic configurations following the typical method used in the literature to evaluate recognizers [3,5,6,121,122,133].

We compare the recognizers in a *user-independent scenario*, *i.e.,* we evaluate the recognition on gestures produced by different users than the templates used to train the recognizers. The 14 basic configurations defined above are further refined depending on two parameters: (1) N, the number of resampling points ($N \in \{4, 8, 16, 32, 64\}$); and (2) T, the number of templates per

gesture chosen from all users to train the recognizers. This second parameter varies according to the dataset: T={1, 2, 4, 8, 16} for SHREC2019 and T={1, 2, 4, 8, 11} for 3DTCGS. Furthermore, *R3D* and *RS* require at least two templates to be instantiated as they cannot compute weights if the number of samples is too small [106]. As such, we defined T={2, 3, 4, 8, 16} (*resp.*, T={2, 3, 4, 8, 11}) for *R3D* and *RS* for the SHREC2019 dataset (*resp.*, the 3DTCGS dataset).

The testing procedure for a recognizer is straightforward. First, one "testing" template is randomly selected for each gesture class from all participants. Then, a training set is generated by randomly choosing T templates for each gesture class for all users. These templates should be different from the templates previously selected for the testing. Finally, the recognizer is trained on the resulting training set. Each time a candidate is positively recognized, an internal counter is incremented. This operation is repeated 100 times for each T. In the end, the number of recognized gestures is averaged to get the recognition rate and format it as a percentage for each N for each recognizer. Overall, we performed 2 (DATASET) × 5 (resampling values N) × 5 (values for T) × 100 (repetitions) × 7 (RECOGNIZER) = 35,000 recognition trials.

4.2 Results

4.2.1 Recognition Rate

Figure 4.2 plots the recognition rates for all recognizers on the SHREC2019 and 3DTCGS datasets with the individual rates obtained in the upper part and the global (aggregated) rates in the bottom part to summarize the results. To be consistent in the presentation style, we systematically use the same type of chart presented in [122], with the same design. Next, we summarize the results for the SHREC2019 dataset. We do not summarize the results for 3DTCGS as they are quite similar. The only major differences compared to SHREC2019 are that the recognition rates increase faster with T and that the global recognition rates are better for *R3D* (τ=74.48% instead of τ=52.51%) and *RS* (τ=68.80% instead of τ=40.82%).

- *$P³+* turns out to be the most accurate recognizer in all conditions (N={4, 8, 16, 32, 64}) in terms of absolute values for all values of T. When N=64, *$P³+* is almost joined by *$Q³*, thus suggesting that $Q exhibits a better accuracy when the amount of templates is increasing. The more points are included for each template, the more accurate *$Q³* becomes as it is the case for $Q [122]. *$P³+* is appropriate for gestures with high sampling, even with a low number of templates (*e.g.*, τ=80.00% with T=4, increasing above 90%).
- *$P³+* is significantly more accurate than all other recognizers. We computed Student's t test for two paired samples since the samples were the same. There was a very highly significant difference in the global recognition rates for *$P³+* ($M$=86.94%, SD=15.00%) and all the other conditions (all with df=2499 and p^{***}<.001):
  $F ($M$=79.36%, SD=18.08%), t=18.74, with a Cohen's d=.37 for a small effect [29];
  $Q³ ($M$=79.36%, SD=18.60%), t=18.67, with a Cohen's d=.37 for a small effect [29];
  FH (M=79.28%, SD=18.51%), t=18.20, with a Cohen's d=.36 for a small effect [29];
  $P³ ($M$=78.10%, SD=19.20%), t=21.13, with a Cohen's d=.42 for a small effect [29];
  RD3 (M=52.51%, SD=32.24%), t=52.35, with a Cohen's d=1.05 for a large effect [29];
  RS (M=40.82%, SD=30.22%), t=73.61, with a Cohen's d=1.47 for a very large effect [102].
  These results suggest that using the *$P³+* recognizer has a positive effect on global recognition rates. This is also confirmed by the values obtained by a one-way ANOVA with Tukey's HSD post-hoc analysis: the between-group analysis also returned a very highly significant effect (SS=11036.5, df=6, F=1444.58, p^{***}<.001).

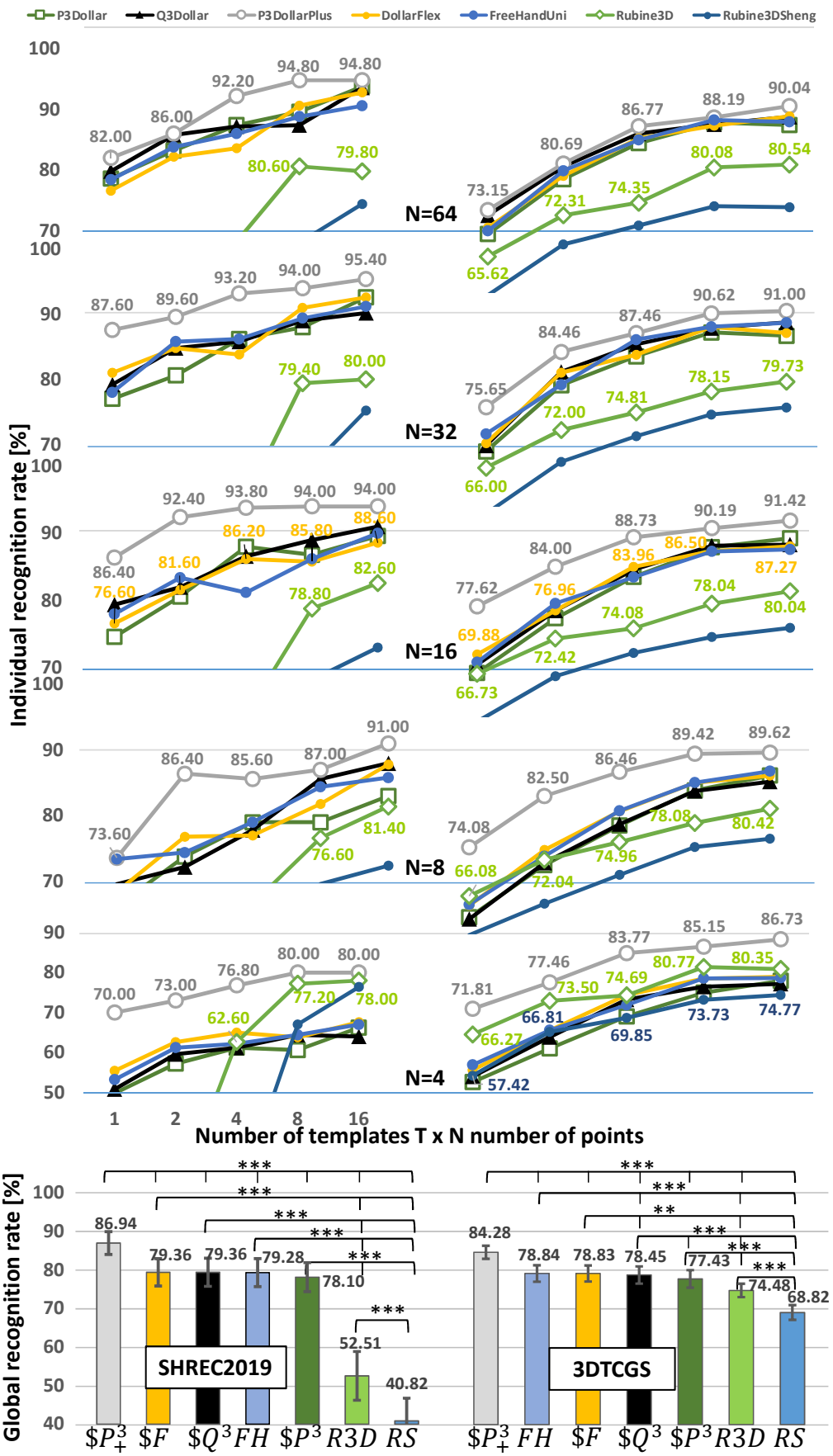


Figure 4.2: Recognition rates of all recognizers for two datasets (left: SHREC2019 [17], right: 3DTCGS [18]): individual rates (top) for N=4, ..., 64 (top) and global rates (bottom) $\forall N$ ($p^{***} \leq .001$). Error bars show a confidence interval with α=.05.

- *$P^3+* could accommodate gestures with low sampling: already with one template on four points (N=4, T=1), it benefits from a reasonable rate of τ=70%, the lowest value for this recognizer in all configuration. When N=4, better to have at least 8 points to reach a rate of τ=80%. When the number of points reaches 8 (N=8), two or more templates are admissible to get a rate of at least (τ=86%). This observation seems to confirm the observation that for $-family members, a sampling of 8 points already gives acceptable results [118]. When $N \geq 16$, one template is already enough and there is no need to go up to 16, even if the best rate is obtained when N=32 with τ=95.40%.
- Apart from *$P^3+*, all other $-like recognizers display a global rate of $\tau \geq 78\%$. Their differences are very light: they remain included in the same envelope with similar rates and progression when $N \nearrow$, $T \nearrow$, and the curves are almost superimposed all the time.
- Although *P^3* returned the lowest global rate of all $-like recognizers, the one-way ANOVA did not return any significant difference between them, but all of them are always significantly more accurate than *RD3* and *RS*.
- Regarding the two feature-oriented recognizers, Rubine3D (R3D) is always more accurate than its counterpart Rubine-Sheng (RS) in all conditions. A Student's t test for two paired samples revealed a very highly significant difference in the global recognition rates for *RD3* (M=52.51, SD=32.20) and *RS* (M=40.82%, SD=30.22%): t=20.16, df=2499, $p^{***}<.001$, with a Cohen's d=.40 representing a small effect. Both exhibit an acceptable rate only when $N \geq 8$. Below this threshold, they are not useful. Surprisingly, when N=4, *RD3* and *RS* outperform other $-like recognizers, apart from *$P^3+*, when $T \geq 4$. After that, they are completely surpassed.
- Rubine-Sheng is the least accurate recognizer in almost all conditions as it is outperformed by all other recognizers tested. Therefore, computing more geometric features for a 3D trajectory does not help much, and keeping the original features in the three planes is enough to outperform Rubine-Sheng.
- If we assume that a human-acceptable rate of $\tau \geq 90\%$, the ideal cases are all when T=16 for all conditions.

4.2.2 Execution Time

We now investigate the recognizers' efficiency based on their execution time (computed as the sum of the pre-processing time plus the recognition time for each candidate) for different configurations. Figure 4.3 plots the execution time of each recognizer on the SHREC2019 and 3DTCGS datasets. We summarize the results for the SHREC2019 dataset:

- Overall, Rubine3D (M=.047, MD=.044, SD=.013) and RubineSheng (M=.017, MD=.015, SD=.006) have the lowest and the most constant times in all configurations. On the contrary, the $-like recognizers are the most time-consuming: *$F* ($M$=.289, MD=.088, SD=.490) is the fastest recognizer among them, followed by *$P^3+* ($M$=.363, MD=.094, SD=.649), FreeHandUni (M=.492, MD=.111, SD=.956), *Q^3* (M=3.005, MD=1.623, SD=3.005) and *P^3* (M=4.798, MD=.668, SD=10.224). This difference suggests that feature-based recognizers (*i.e., RD3* and *RS*) and template-based ones (*i.e., P^3*, *P^3+*, *Q^3*, *$F*, and FreeHandUni) are not influenced by the same parameters.
- RubineSheng is the fastest recognizer in most N and T conditions, except in the case where N=4 and a reduced number of templates are provided to the recognizer ($T \leq 4$).
- The times of all $-like recognizers increase while the number of points and the number of templates grow.

- *$Q*^{3} is the slowest recognizer for N=4, 8. Other curves are under the *$Q*^{3} one but from N=16, *$P*^{3} execution times increase quickly and exceed the execution time of *$Q*^{3} for T=16 ($t_{\$P^3}$=2.196 ms, $t_{\$Q^3}$=1.853 ms) and it occurs even quicker for N=32 and N=64, where *$P*^{3} exceeds *$Q*^{3} immediately after T=4, resp., T=2.
- *RS* is three-time faster than *RD3*: the ratio of their averaged execution times is $\frac{t_R S}{t_R 3D}$=2.82. This ratio makes sense considering that *RD3* computes 39 features of the candidate (3 times 13 features for the three planes XY, YZ, and ZX) while RS computes 16 features once.
- For these feature-oriented recognizers, the candidate is pre-processed, its features are extracted and multiplied by pre-computed weights of each class, then summed. The candidate's class is designated by the larger sum resulting from this calculation, which confirms the constant time for the recognition, only influenced by the number of classes in the dataset.
- The *$F*, *$P*^{3}+, and FreeHandUni have short execution times compared to other $-like recognizers. *$F* and *$P*^{3}+ are better than FreeHandUni ones, this variation is caused by the early abandoning for the two recognizers.
- Knowing that 100 ms is the limit for a user to feel that a system is operating in real-time [87], the execution times for all the conditions for this dataset are fortunately under this limit.

The overall picture remains the same for the 3DTCGS dataset: the execution times increase for all the recognizers and conditions, which pushes the *$P*^{3} over the limit of 100 ms. This increase is due to the higher number of classes and samples per class in the dataset (26 classes compared with 5 gestures for SHREC2019). We observe a rapid increase of the *$P*^{3} curve which surpasses the *$Q*^{3} just after T=4 for N=8.

4.3 Conclusion

This chapter showed that, in addition to the development of gesture-based UIs, the framework can also serve more general purposes such as comparing gesture recognizers. This benchmarking compared the accuracy and execution time of seven recognizers (*i.e.,* three that were described in Section 3.2.3 and four that were integrated into the framework for this specific application) on two challenging gesture sets that were used in gesture recognition contests. The modular architecture of the framework makes it easy to vary the benchmarking configuration, by adding or removing recognizers, changing datasets, or selecting a subset of gestures from a dataset. It is thus possible to obtain other similar configurations for testing a particular recognizer against various gesture sets, a particular gesture set across a series of recognizers, and for benchmarking several recognizers against several gesture sets.

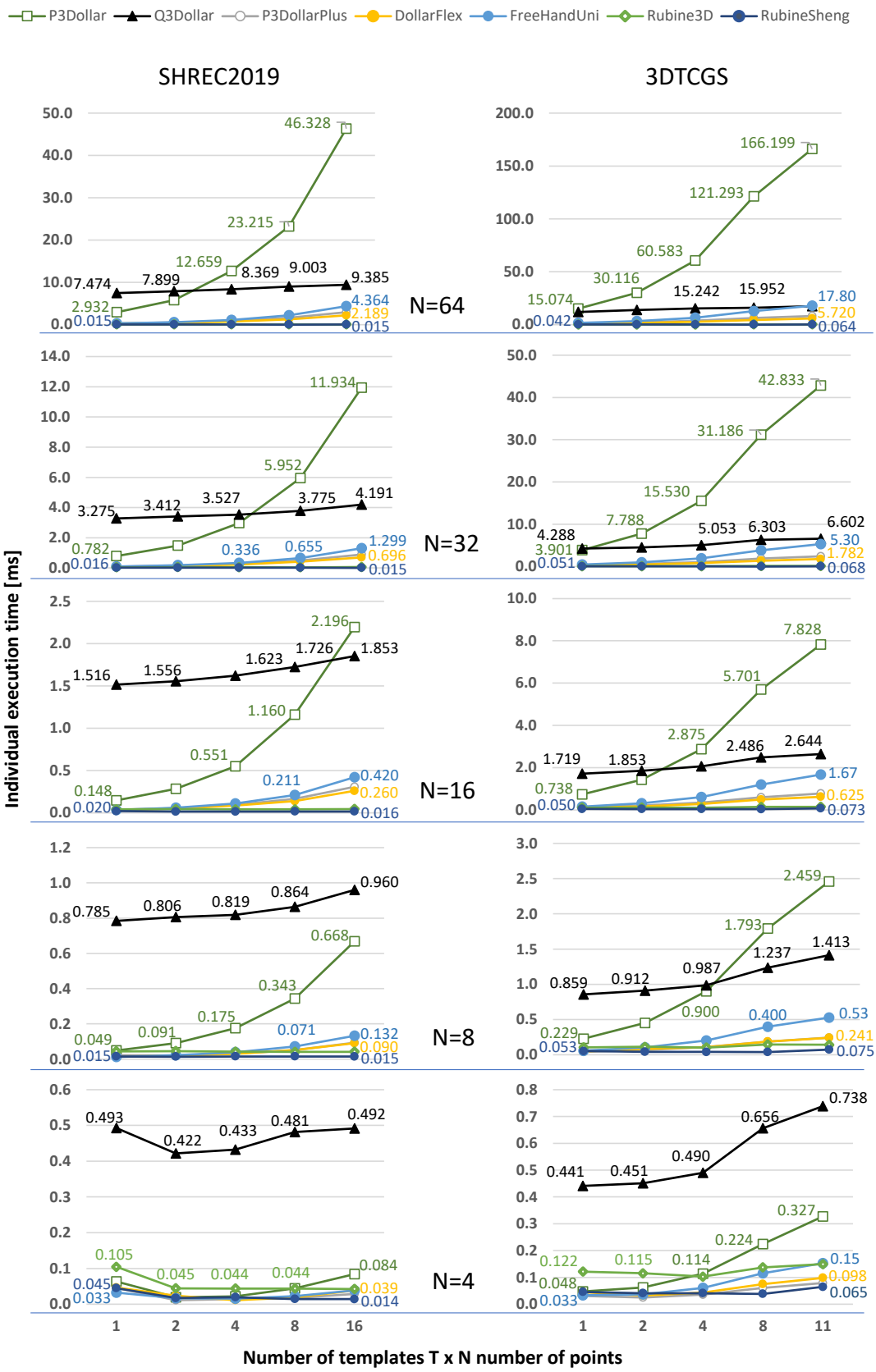


Figure 4.3: Execution times of all recognizers for two datasets (left: SHREC2019 [17], right: 3DTCGS [18]): individual times for N=4, ..., 64.

Chapter 5

The Large User Interface Project

This chapter presents a third case study of using the framework for engineering gesture-based UIs to a large-size application: the Large User Interface project. This chapter is composed of four parts. Section 5.1 first gives an overview of the LUI project, the rationale behind it, and its current state. Then, Section 5.2 describes how we designed a suitable gesture set by applying a gesture elicitation study for the application while Section 5.3 is dedicated to finding the most appropriate recognizers for this gesture set. Finally, Section 5.4 details how we used the framework to implement a complete gesture-based UI for the LUI project.

5.1 Project Overview

The Large User Interface (LUI) project [89] managed by The MIT Media Lab[1] is aimed at allowing people to naturally interact by gesture to manipulate multimedia contents on a large display, such as a wall display or a tabletop. As multimedia contents, such as pictures, videos, and maps, are probably the most common elements that are widely manipulated by the largest amount of users [36], an intuitive and accessible gesture-based UI could quickly be adopted by a significant part of the population. This project is mainly developed by Vik Parthiban at MIT as such: "LUI is a scalable, multimodal web-interface that uses a custom framework of non-discrete, free-handed gestures and voice to control modular applications with a single stereo-camera and voice assistant".

5.1.1 Rationale

As large displays have become more and more accessible in the last few years, input methods such as the classic keyboard-and-mouse combo or TV remotes have become increasingly ill-suited to the task. In this context, the objective of LUI's creators is to provide a UI that is easy to control through both gestures and voice commands, while being "easily installed and deployed on any large screen surface" [89]. When designing the system, they defined the three following design guidelines:

1. *Accessible*: the system should be both cheap and easy to set up and to use.
2. *Extensible*: apps should be simple to add, modify, and remove from the system.

[1]This work is part of a larger collaboration project between MIT and UCLouvain in the context of an MIT-UCL MISTI Fund "Large User Interface" project lead by Prof. Pattie Maes on the MIT side and by Prof. Jean Vanderdonckt on the UCLouvain side, under contract #200278.

3. *Non-discrete*: to avoid latency, the system should support non-discrete methods of interaction.

5.1.2 Current State

The system is built using the popular ReactJS framework to provide a reactive UI that can be easily modified. Each application, such as the *Photos* or the *Videos* applications, consists of one separate module. As such, writing new apps is easy as developers only have to focus on one module, which contains all of their application's code. In its current version, the LUI project is composed of only four major modules, including two applications, forming the functional core of the LUI project:

- *Welcome screen* (Figure 5.1a): the entry point of the LUI project. From this screen, one can only go to the app menu.
- *App menu* (Figure 5.1b): a list of all the applications. Clicking on an application opens it.
- *Videos app* (Figures 5.1c and 5.1d): a list of YouTube videos, organized into multiple pages. Each video can be selected and viewed in full-screen. From the full-screen mode, users can play/pause the video, or go to the previous and next ones in the list.
- *Photos app* (Figures 5.1e and 5.1f): a list of pictures, organized into multiple pages. As in the *Videos* app, each photo can be selected and viewed in full-screen, and one can browse through the photos directly from the full-screen mode.

Four other applications are also available, but they have been disabled in the latest version of the project. For this reason, we do not describe them here.

The hypothesis of this chapter consists of the availability of the functional core of the aforementioned modules, for which a gesture UI should be developed based on the LMC. The JavaScript code corresponding to this functional core has been made available by Vik Parthiban in the context of the MISTI project.

5.2 Designing an Appropriate Gesture Set

Designing a good gestural interface requires some insight regarding potential users' preferred gestures for interacting with the application. A great way of gathering such insight is to perform elicitation studies, *i.e.,* experiments in which users are asked to associate their preferred gestures to a series of actions (called referents) in a controlled environment.

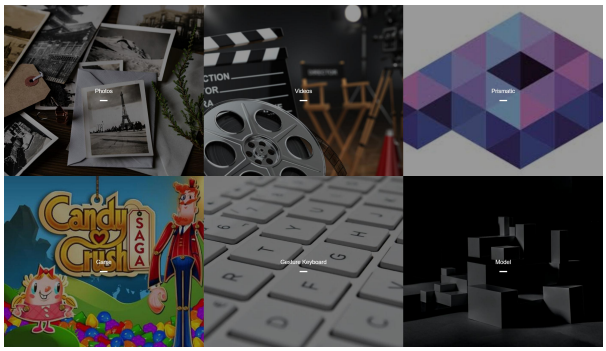
The remainder of this section is divided into five parts. Section 5.2.1 briefly describes two preliminary gesture elicitation studies (GESs) that were conducted to explore user-defined gestures and highlights their shortcomings. Then, Section 5.2.2 details the updated procedure used for a more recent, updated, GES and Section 5.2.3 reports its results. Section 5.2.4 explains how, based on these results, we created a gesture set for the LUI project. Finally, Section 5.2.5 classifies the gestures of the new gesture set according to Aigner *et al.*'s taxonomy [2].

5.2.1 Preliminary Gesture Elicitation Studies

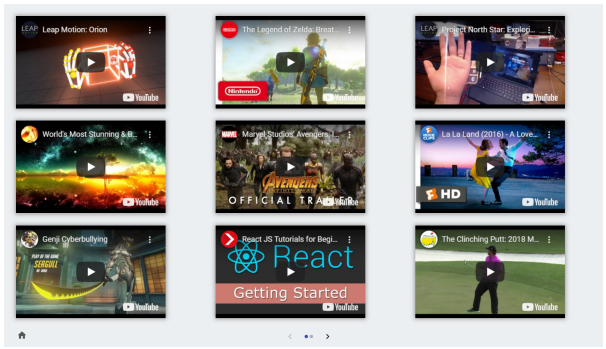
The LMC comes with five system-defined gestures, *i.e.,* "swipe right/left/up/down", and "Explode", that are accessible through a programmatic way that does not satisfy our software quality properties. These system-defined gestures, as well as designer-defined gestures, are not always easy to remember by end users [84], who often prefer to rely on their own user-defined gestures. To explore which gestures would be preferred by LUI end-users, two preliminary GESs



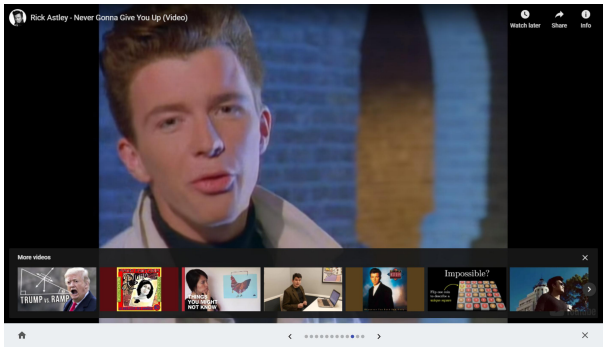
(a) Welcome screen



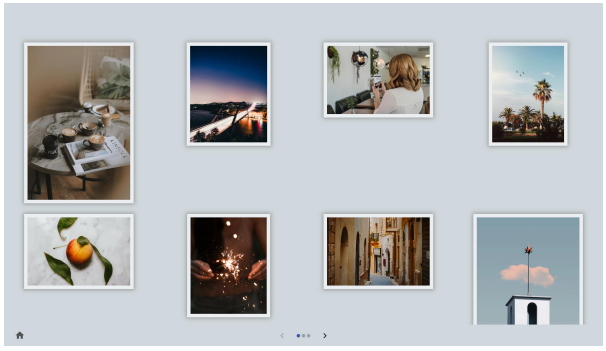
(b) App menu



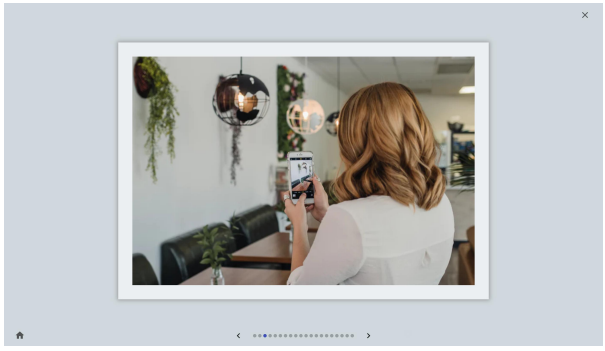
(c) Videos app (list view)



(d) Videos app (full-screen)



(e) Photos app (list view)



(f) Photos app (full-screen)

Figure 5.1: Screenshots of the LUI application.

were conducted in 2019 following the general methodology (*e.g.,* [32,42,124,131,133]) to collect users' preferences for mid-air gesture input for browsing through respectively photos and videos.

Gesture Elicitation Studies [32,124,132] are popular for investigating users' preferences for gesture input in a variety of contexts of use [99,121,125,132]. For instance, Wobbrock et al. [132] reported users' preferences for multi-touch input on interactive tabletops. Zaiţi et al. [125,137] addressed mid-air gesture input to control a TV set. The result consists of a characterization of users' gesture input behaviors with valuable information for designers, practitioners, and end-users regarding the consensus among participants (which is computed as an *agreement* or *coagreement rates* [32,124,132]), most frequent (thus, generalizable across users) gestures for a specified task, and insights into users' conceptual models for performing tasks. The most recent formalization of the elicitation method proposed both repeated measures [124] and between-subjects [32] experimental designs. See [126] for a survey of GESs.

In the rest of this section, we briefly describe the setup of the two preliminary GESs and highlight some of their shortcomings.

Setup

In both experiments, participants were installed at a desk and faced a laptop, which showed them referents. For each of these referents, they had to propose a suitable gesture, *i.e.,* a gesture that fits the referent well while being easy to perform and remember. There were no constraints on the participants' choices of gestures, with the exception that a given gesture could not be assigned to more than one referent. The order of the referents was randomized per participant.

Because we did not have a Kinect or LMC at our disposal for the experiment, participants were asked to perform their gestures in mid-air in front of the laptop, as if it was able to register their hand gestures (see Figure 5.2). Participants' fine motor skills and creativity levels were evaluated before the experiment, and a short questionnaire was administered after the experiment to give us an insight into the system's user satisfaction.

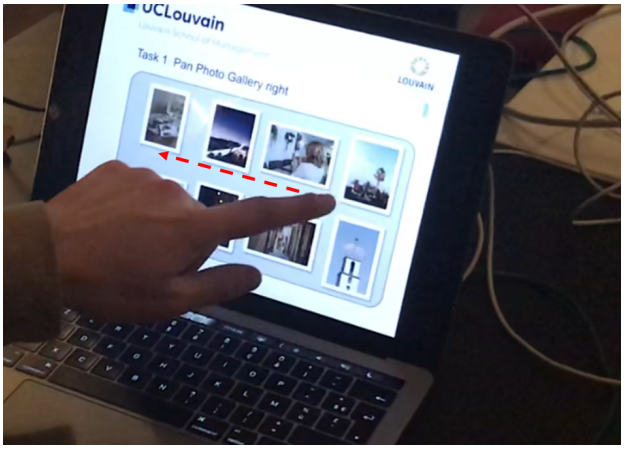


Figure 5.2: Snapshot from the GES showing a participant performing a mid-air gesture.

Observations and Possible Issues

When analyzing the results of both GESs (details are available in Appendix B), we identified the following shortcomings:

- *Unrealistic setup*: the experiment's setup may have impacted participants' choice of gestures. For instance, in the *photo-browsing* experiment, participants were situated very close to the screen with the visualization of the referents, to the point where some of them physically touched it while performing gestures. This may have encouraged them to propose more "touch-oriented" gestures.
- *Unique gestures requirement*: requiring participants to assign a different gesture for each referent sometimes forced them to propose less-than-ideal gestures, *e.g.,* if they had already assigned their preferred gesture to another referent.
- *Unclear visual indications*: some visual indications of the referents did not provide a clear idea of what the meaning of the represented action was. Designing clear visualizations for each referent could help participants propose more suitable gestures and speed up the elicitation process.
- *Too many referents*: some participants complained about the high number of referents (*resp.*, 27 and 31 for the *photo-browsing* and *video-browsing* experiments), which greatly increased the duration of the experiment. Reducing the number of referents could ensure that participants stay focused throughout the experiment.

- *Too complex referents*: some referents referred to complex actions, which combined multiple concepts at once while providing insufficient details (*e.g.,* "search photos within the gallery by criteria", implies both a "search" and a "select a criteria" action). Such referents should be clarified and/or divided into multiple smaller referents.

5.2.2 New Gesture Elicitation Study

This section details a new GES that was performed in collaboration with Nathan Magrofuoco. Due to the confinement measures imposed by the Belgian government during the COVID-19 outbreak, we reached a smaller number of participants than expected.

Addressing the Shortcomings of the Previous Experiments

This new experiment mostly follows the same protocol, inspired by Gheran *et al.* [42], as the two preliminary GESs mentioned above. However, it was slightly modified in an attempt to address their shortcomings. First, both the number of referents and their complexity have been drastically reduced. Second, the setup of the new experiment has been modified to be more realistic. The screen is now further away from the participants, and the LMC can record their hand motions. Finally, participants are encouraged but not required to find a unique gesture for each referent.

Participants

Twenty-three (23) participants (6 female), aged between 18 and 56 years old (M=27.1, SD=10.2 years), volunteered for the study. Their occupations include students and employees, mostly in IT, art, and engineering. All but two of the participants had never used an LMC, but all of them regularly used a computer. Only one participant did not frequently use a smartphone. 95.7% of the participants (22/23) were right-handed.

Apparatus

Compared to the two previous elicitation studies, the setup is much closer to the expected context of use of the system. The experiment relied on three main devices: the LMC to capture the mid-air gestures proposed by the participants, a smartphone to record their touch gesture propositions, and a large screen to display the referents. A laptop computer was also used to control the display and record the gestures captured by the LMC using a custom tool that we developed (see Appendix C).

Design

The experiment has three independent variables. Two of them are within-subjects variables: the REFERENT and the INTERACTION METHOD.

The REFERENT is a nominal variable with 18 conditions. Each condition represents a common task to perform in multimedia environments, such as in a music player, a video gallery or a PowerPoint presentation: (1) previous; (2) next; (3) enable fullscreen; (4) disable fullscreen; (5) zoom in; (6) zoom out; (7) volume up; (8) volume down; (9) rotate 90° clockwise; (10) rotate 90° anti-clockwise; (11) play; (12) pause; (13) like; (14) dislike; (15) fast forward 5 seconds; (16) rewind 5 seconds; (17) enable subtitles; and (18) disable subtitles. The number of conditions has been reduced compared to the previous experiments by combining similar referents in different contexts and dropping the most uncommon actions.

The INTERACTION METHOD is a nominal variable with 2 conditions: (1) touch gesture; and (2) mid-air gesture.

The last variable, the PROPOSITION SEQUENCE, is between-subjects. It has 2 conditions: (1) for each referent, the participant first proposes a touch gesture and then a mid-air gesture; and (2) for each referent, the participant first proposes a mid-air gesture and then a touch gesture.

In the context of this thesis, we will only focus on participants' mid-air gesture propositions.

Task

Participants were installed at a desk and faced a large screen situated at a few meters on the opposite side of the desk. The LMC was positioned in front of them, and a smartphone was placed on their right. Each participant was presented with a series of referents in a randomized order. For each one of these referents, participants were asked to propose both a touch and a mid-air gesture that fit the referent, in an order determined randomly at the start of the experiment. Participants' thinking time was measured for each touch and mid-air gesture proposition. There were no constraints on the participants' choices of gestures. After each gesture proposition, the participants estimated the goodness-of-fit of their proposition on a Likert scale from 1 (very unsatisfied) to 5 (very satisfied).

Measures

We employed the following measures, taken from Gheran *et al.* [42], to evaluate and understand users' preferences for gestures detected by the LMC:

1. We computed agreement rates (AR) for each REFERENT using the formula of Vatavu and Wobbrock [124] (p. 1327), as follows:

$$AR(r) = \frac{\sum_{i<j} \delta_{i,j}}{n \cdot (n-1)/2} \tag{5.1}$$

   where n is the number of participants from which gestures are elicited, and $\delta_{i,j}$ evaluates to 1 if the i-th and j-th participants are in agreement over referent r and to 0 otherwise.
2. THINKING-TIME measures the time, in seconds, needed by participants to propose a gesture for a given referent.
3. GOODNESS-OF-FIT represents participants' subjective assessment, as a rating between 1 and 5, of their confidence about how well the proposed gestures fit the referents.

5.2.3 Results

We collected a total number of 414 mid-air gesture proposals from 23 (participants) × 18 (referents) conditions. We then clustered these gesture propositions into 102 groups of similar gestures. For this clustering, we distinguished between gestures according to their direction (*i.e.,* the same gestures performed in different directions are considered different) but grouped similar gestures performed with different poses together (*e.g.,* swiping with a flat hand is considered the same as swiping while pointing the index finger). In the rest of this section, we analyze the consensus between participants' gesture proposals, as well as its relation with thinking time and goodness of fit.

Referent	AR	Most frequent	Second most frequent
1. *Previous*	.498	Swipe right	Swipe left
2. *Next*	.498	Swipe left	Swipe right
3. *Enable fullscreen*	.273	Bring hands further apart	Pinch out
4. *Disable fullscreen*	.146	Bring hands closer	Pinch in
5. *Zoom in*	.320	Pinch out	Bring hands further apart
6. *Zoom out*	.300	Pinch in	Bring hands closer
7. *Volume up*	.478	Swipe up	Rotate a knob clockwise
8. *Volume down*	.316	Swipe down	Rotate a knob anti-clockwise
9. *Rotate 90° clockwise*	.830	Rotate a knob clockwise	N/A
10. *Rotate 90° anti-clockwise*	.830	Rotate a knob anti-clockwise	N/A
11. *Play*	.190	Tap with the index finger	Draw the "play" symbol
12. *Pause*	.166	Tap with the index finger	Swipe down with two fingers
13. *Like*	.751	Thumbs up	N/A
14. *Dislike*	.751	Thumbs down	N/A
15. *Fast-forward 5 seconds*	.087	Swipe right twice	Swipe left
16. *Rewind 5 seconds*	.095	Swipe right	Swipe left twice
17. *Enable subtitles*	.032	Swipe up	Swipe forward
18. *Disable subtitles*	.028	Swipe down	Swipe backward

Referents with agreement rates above average are highlighted in blue.

Table 5.1: First and second most frequent gesture proposals for each referent.

Consensus Between Proposed Gestures

Figure D.1 shows the agreement rates obtained for each Referent. Overall, agreement rates are quite high, between .028 and .83 (M=.366, SD=.268). These results fit within the reported agreement rates in the literature of gesture elicitation; see Vatavu and Wobbrock [124] (p. 1332) that summarize agreement rates of 18 studies. According to the recommendations of Vatavu and Wobbrock [124] to interpret the magnitudes of agreement rates, most of our results fall inside the high consensus (.300 − .500) category.

These results were confirmed by Kendall's coefficient of concordance:[2] W=.564 ($\chi^2(26)$=220.454, p<.0001). As Kendall's coefficient is related to the average of Spearman correlation coefficients between pairs of rankings [51] (p. 276), we can interpret the magnitude of its effect as large (*i.e.,* close to .500) according to Cohen's suggested limits for interpreting effect sizes.

Gestures' Goodness of Fit

Participants rated their gesture proposals with numbers from 1 (poor fit) to 5 (excellent fit) to denote their confidence in the goodness of fit of their proposals. Overall, participants were satisfied with their gesture propositions as shown by the high average goodness of fit (M=4.21, SD=0.88). Goodness-of-Fit correlated significantly with the Agreement-Rate (Pearson's $r_{(N=18)}$=.711, R^2=.505, p=.00094): referents that reached high agreement rates were assigned gestures that were rated a good fit. Figure D.2 shows this relationship between the goodness of fit and agreement rates.

[2]Kendall's W is a normalization of the Friedman statistic used to assess the agreement between multiple raters with a number ranging between 0 (no agreement at all) and 1 (perfect agreement).

Thinking Time

The average thinking time was usually quite low (M=6.15 s, SD=9.12 s). Figure D.3 shows the relationship between thinking times and agreement rates per Referent. Thinking-Time correlated significantly with Agreement-Rate (Pearson's $r_{(N=18)} = -.620$, R^2=.384, p=.006): the agreement rate decreased for longer thinking times. A few referents showed considerably higher than average thinking times: disable subtitles, enable subtitles, and skip backward 5 seconds. The reason behind this could be that these referents represent tasks that participants are less likely to perform regularly on their devices.

5.2.4 Choice of Gestures

In the previous sections, we elicited users' preferred gestures for a series of actions related to the LUI project. However, one would be ill-advised to use these gestures directly, as gestures with the highest agreement rate may not always be the most appropriate ones for a given system. In this section, we assign an appropriate gesture to each action, by taking into account both the results of the elicitation study as well as possible hardware and software limitations.

Table E.1 presents, for each action, its assigned gesture, as well as a justification for this choice. In general, we tried to assign symmetric gestures to symmetric actions (*e.g.,* the "flick left" and "flick right" gestures to resp. the "next" and "previous" referents) in an attempt to increase the memorability of the gestures. We also avoided using the same gesture for two different actions that could occur in the same context to avoid any conflict between them.

The "fast-forward" and "rewind" gestures could be swapped based on users' preferences. Indeed, some users may prefer the swiping motion to correspond to them physically moving the video forward or backward. In this case, "swipe left" would be associated with the "fast-forward" action. On the other hand, others may associate this same swiping motion to moving a pointer on a timeline. In that case, "swipe left" would be associated with "rewind". Here, we chose to keep the first option, as it seemed more popular.

Figure E.1 shows, for each referent, a representation of its assigned gesture. Please note that, due to both a low consensus among the participants and their comparatively low interest compared to the other referents, we chose to discard two of the referents present in our latest GES: "enable subtitles" and "disable subtitles".

5.2.5 Gesture Taxonomy

We now characterize the gestures from our newly defined gesture set, following Aigner *et al.*'s taxonomy [2]. This step will prove extremely useful later to configure the framework and determine the most appropriate gesture recognition algorithms. According to Aigner *et al.*, gestures can be divided into five main types [2]:

- *Pointing*: gestures that indicate objects and directions.
- *Semaphoric*: hand poses and movements used to convey a specific meaning (which is often unrelated to the gesture). They can be further divided into three categories: static, when the meaning is entirely conveyed by the hand pose; dynamic, when the temporality of the gesture is important; and strokes, which are simpler as they only consist of single stroke hand motions.
- *Pantomimic*: gestures that imitate a real-life task, such as opening an invisible bottle.
- *Iconic*: gestures that communicate information about objects or entities (*e.g.,* sizes, shapes, or paths). They can be further divided into two categories: static when all the information

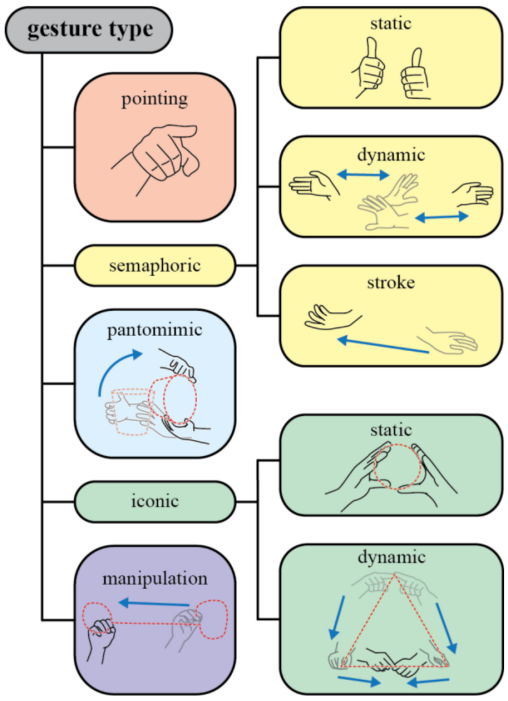


Figure 5.3: Aigner *et al.*'s classification of gestures [2].

Action	Gesture	Category
Previous	Flick right	Stroke semaphoric
Next	Flick left	Stroke semaphoric
Enable fullscreen	Tap	Pointing
Disable fullscreen	Flick up	Stroke semaphoric
Zoom in	Pinch out	Manipulation
Zoom out	Pinch in	Manipulation
Volume up	Swipe up	Manipulation
Volume down	Swipe down	Manipulation
Rotate clockwise	Rotate a knob clockwise	Manipulation
Rotate anti-clockwise	Rotate a knob anti-clockwise	Manipulation
Play	Tap	Stroke semaphoric
Pause	Tap	Stroke semaphoric
Like	Thumbs up	Static semaphoric
Dislike	Thumbs down	Static semaphoric
Fast-forward	Swipe left	Manipulation
Rewind	Swipe right	Manipulation

Table 5.2: Gesture classification according to Aigner *et al.*'s taxonomy.

is conveyed by a hand pose; and dynamic, such as movements that describe paths or shapes.

- *Manipulation*: gestures used to manipulate objects in a short feedback loop. The effect of the gesture on the object is perceived while the user is performing it, with only a very short latency.

Table 5.2 classifies each of the 16 selected gestures according to Aigner *et al.*'s taxonomy. One can observe that, depending on the context, the *tap* gesture may belong to two different categories. In the case of the *enabling fullscreen* action, *tap* also acts as a selection method, as users first need to hover their finger above the correct element before executing the gesture. In this context, it belongs to the *pointing* category. For the other actions, only the stroke motion is important, and *tap* is thus *stroke metaphoric*.

Eight of the 16 gestures are classified in the *manipulation* category, as they are used to directly modify or interact with content. For instance, the size of an image should change proportionally to the thumb-index distance when the user is performing a *pinch in* or *pinch out* gesture. Using the gesture recognition framework, we can support these gestures, as well as the two *static semaphoric* gestures, by combining a classifier and an analyzer module.

Finally, five gestures are considered as *stroke semaphoric*, and one as *pointing*. Recognizing them could prove tricky, as the framework first needs to correctly separate them from parasitic hand motions (*i.e.,* perform gesture segmentation). Thus, in addition to a recognizer module, we also need to use a segmenter module. Furthermore, as strokes mostly differ in their direction, we should avoid using direction-independent recognizers.

5.3 Finding the Most Appropriate Gesture Recognizers

This section is aimed at determining the most appropriate recognizers for the gesture set that we defined in the previous section. It first describes the testing procedure and then summarizes the results.

5.3.1 Procedure

We ran the benchmark on a computer running Windows 10 Home (version 1909), equipped with a Core i7-8750H CPU running at 2.20 GHz and 16 GB of DDR4-2667MHz RAM. We tested the dynamic and static gesture recognizers separately. All the recognizers are detailed in Chapter 3 but Table 5.3 summarizes their invariance properties.

	Type of invariance			
Recognizer	**Position**	**Scale**	**Rotation**	**Direction**
3 cent	✔	✔	✘	✘
$P^3	✔	✔	✘	✔
$P^3+	✔	✔	✘	✔
$P^3+X	✔	✔	✘	✘/✔
$Q^3	✔	✔	✘	✔
Jackknife	✔	✔	✘	✘
GPSDa	✔	✔	✔	✔

Table 5.3: Invariance properties of the recognizers.

The dynamic gesture recognizers were tested on pre-segmented gestures acquired from four participants (one of them is the author of this thesis). Each participant repeated the gestures four times, giving a total of 16 templates per gesture class. There are four gesture classes, corresponding to those needed for the LUI application. We fixed N, the number of resampling points to 16, which is a reasonable value for the type of algorithms tested here [118]. There are two parameters: (1) $A \in \{1, 6\}$, the number of articulations (Figure 5.4); and (2) $T \in \{1, 2, 4, 8\}$, the number of training templates. Testing a recognizer is straightforward. For each value of A and T, and for each gesture class, we repeat the three following steps 1000 times: (1) select a random testing template from the gesture set; (2) train the recognizer using T randomly selected templates produced by different users than the testing template (user-independent testing); and (3) attempt to recognize the testing template.

We tested the pose classifiers on a custom gesture set containing five types of poses acquired from one participant (the author of this thesis). Each pose was performed in various orientations and scales, which results in more than 1000 templates per class. There are two parameters: (1) $A \in \{6, 16\}$, the number of articulations (see Figure 5.4); and (2) $T \in \{1, 2, 4, 8, 16, 32, 64\}$, the number of training templates. Please note that we fixed the a parameter of *GPSDa* to 0.70. The testing process is similar to the one described above, except for the second step. For each value of A and T, and for each gesture class, we repeat the three following steps 1000 times: (1) select a random testing template from the gesture set; (2) train the recognizer using T randomly selected templates, produced by the same user as the testing template (user-dependent); and (3) attempt to recognize the testing template.

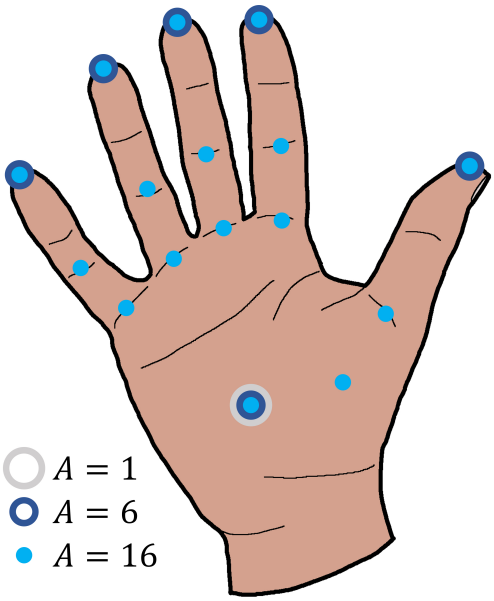


Figure 5.4: Articulations used in the benchmarking.

5.3.2 Results

We first summarize the results for dynamic gesture recognizers. The recognition rates of the recognizers for each gesture class are condensed in Table 5.4.

	Gesture class				Global accuracy		
Recognizer	**Flick up**	**Flick left**	**Flick right**	**Tap**	**Mean**	**SD**	**Time**
3 cent (A=1)	100.0%	96.6%	100.0%	100.0%	99.2%	1.7%	1.05ms
\$P^3 (A=1)	100.0%	67.8%	69.6%	100.0%	84.4%	18.1%	0.41ms
\$P^3 (A=6)	95.6%	93.0%	77.5%	99.8%	91.5%	9.7%	0.46ms
\$P^3+ (A=1)	100.0%	61.3%	53.4%	100.0%	78.7%	24.8%	0.05ms
\$P^3+ (A=6)	99.9%	86.5%	78.2%	100.0%	91.2%	10.7%	0.09ms
\$P^3+X (A=1)	100.0%	99.4%	99.5%	100.0%	99.7%	0.3%	0.06 ms
\$P^3+X (A=6)	99.8%	86.8%	78.6%	100.0%	91.3%	10.5%	0.09ms
\$Q^3 (A=1)	100.0%	63.9%	37.4%	100.0%	75.3%	30.5%	0.88ms
\$Q^3 (A=6)	100.0%	82.4%	72.5%	100.0%	88.7%	13.6%	0.93ms
Jackknife (A=1)	100.0%	100.0%	100.0%	100.0%	100.0%	0.0%	0.05ms
Jackknife (A=6)	100.0%	100.0%	100.0%	100.0%	100.0%	0.0%	0.11ms

Recognizers with a global accuracy higher than 98% are highlighted in blue.

Table 5.4: Recognition rates of each dynamic recognizer, for T=8.

- The highest recognition rates are achieved with 8 training templates (T=8) but all recognizers reach at least 75% of their best recognition rate with only one template. *Jackknife* (A=1 and A=6) and *3 cent* even manage to reach more than 95% accuracy, at respectively 100.0% and 95.6%.
- With only one training template (T=1), *Jackknife* (A=1 and A=6) achieves 100% accuracy over all gesture classes while being as fast as *\$P^3+* and *\$P^3+X*.
- Direction-independent recognizers are not suited for the LUI application, as our benchmark confirms that they cannot correctly differentiate between the “swipe left” and “swipe right”

gestures (see Figure 5.5a). However, we noted that increasing the number of articulations generally improves their recognition rates for these symmetric gestures.

- On this gesture set, *$P³* is slightly more accurate than *$P³+* for both A=1 and A=6. However, *$P³+* executes between 5 and 8 times faster.
- *$P³+X*'s partial direction-invariance does not provide any benefits over *$P³+* when A=6. However, it manages a >20% gain in accuracy when A=1. Figure 5.5 shows the confusion matrices of *$P³+* and *$P³+X*.

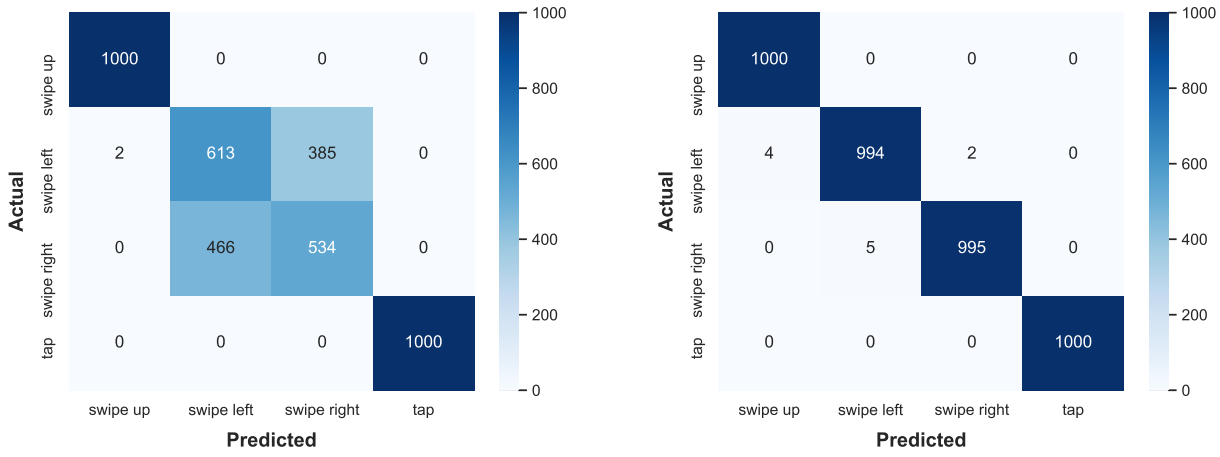
Then, we summarize the results for static gesture recognizers (or pose classifiers). Their recognition rates for each gesture class are condensed in Table 5.5.

- The accuracy of *$P³+* decreases when going from 6 to 16 articulations, while the opposite is true for *GPSDa*.
- *GPSDa*'s execution time is comparable to *$P³+* with 8 training templates. However, it becomes faster as the number of templates increases. With 64 templates, *GPSDa* is twice as fast as *$P³+*.
- Both *$P³+* (6 articulations) and *GPSDa* (16 articulations) reach >90% accuracy with only 8 training templates.
- With 6 articulations and 64 training templates, *$P³+* reaches 99.6% accuracy, while *GPSDa*'s accuracy is lower at 97.1%.
- With 16 articulations and 64 training templates, both recognizers reach >98% accuracy with (*resp.* 98.7% and 99.3% for *$P³+* and *GPSDa*).

	Gesture class					Global accuracy		
Recognizer	**Flat**	**Point**	**Pinch**	**Grab**	**Thumb**	**Mean**	**SD**	**Time**
$P³+ ($A$=6)	98.9%	100.0%	99.5%	99.6%	100.0%	99.6%	0.5%	0.07ms
$P³+ ($A$=16)	99.5%	99.4%	99.3%	98.3%	100.0%	99.3%	0.6%	0.35ms
GPSDa (A=6)	98.6%	97.8%	96.2%	97.3%	95.6%	97.1%	1.21%	0.02ms
GPSDa (A=16)	99.7%	98.3%	97.4%	99.7%	98.4%	98.7%	1.0%	0.18ms

Recognizers with a global accuracy higher than 98% are highlighted in blue.

Table 5.5: Recognition rates of each static recognizer, for T=64.



(a) *$P³+* ($A$=1) (b) *$P³+X* (A=1)

Figure 5.5: Confusion matrices for *$P³+* and *$P³+X* (N=8).

5.4 Implementation

In the previous sections, we designed and characterized a gesture set for the LUI project. In this section, we describe how we configured the framework to suit the needs of this application. Then, we detail how we modified LUI to incorporate the gesture recognition framework.

5.4.1 Framework Configuration

Figure 5.6 gives an overview of the configuration of the framework. Next, we justify the most important configuration choices.

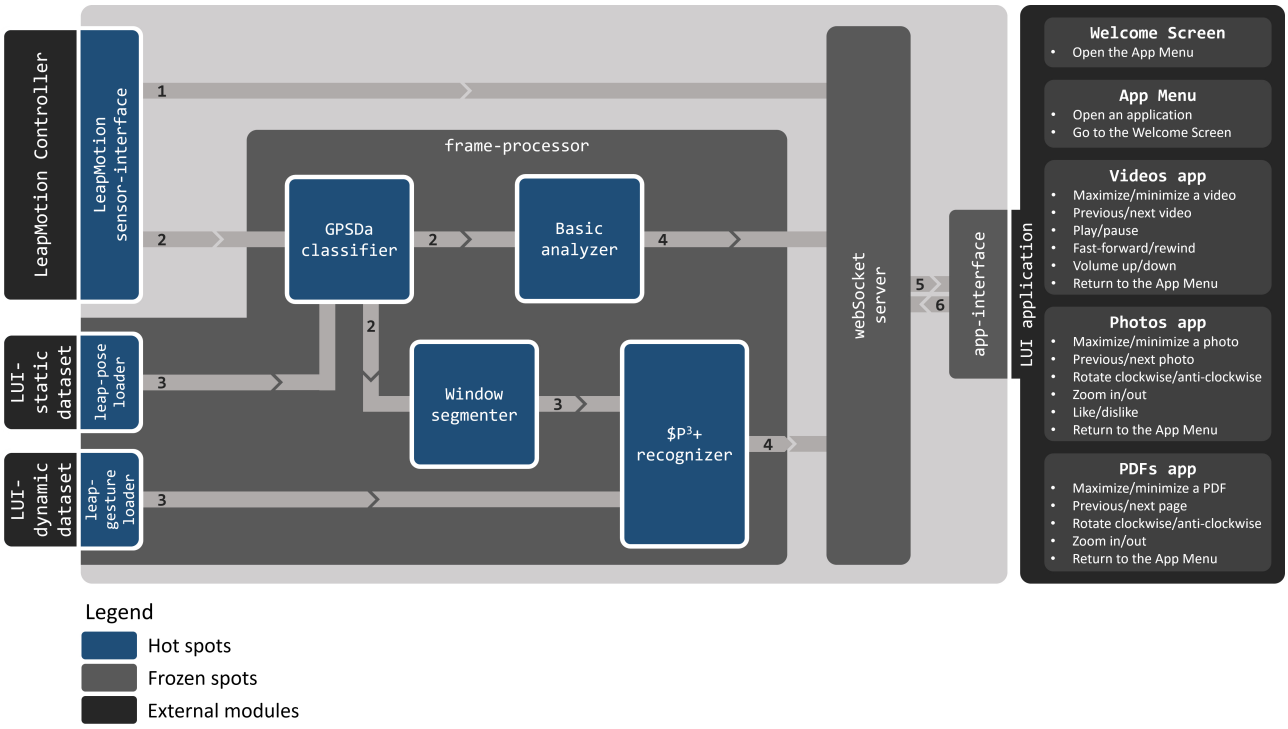


Figure 5.6: Configuration of the framework for the LUI application.

Classifier

We selected the *GPSDa* classifier with 16 articulations and A=0.80 over *$P³+* for two reasons: (1) thanks to its rotation invariance it seems to more accurately classify hand poses under bad conditions (*e.g.,* in case of self-occlusion), especially for the *rotate (anti-)clockwise* actions; and (2) it executes faster, which allows us to train it using more templates and decreases latency. It is trained using 100 random templates for each one of the five poses (flat, point, pinch, grab, and thumb).

Recognizer

From the results in Section 5.3, we selected the *Jackknife* recognizer. It is both fast and accurate, which allows us to keep latency at a minimum with no trade-off on recognition accuracy. We use the window-segmenter configured with a window of 300ms for gesture segmentation and train the recognizer with 16 randomly selected templates per gesture.

Miscellaneous

To match the screen refresh rate of most devices and reduce CPU usage, we fixed the frame-rate of the sensor to $60fps$.

5.4.2 Modifications to the Project

Figure 5.7 shows the gestures that we have implemented in each module of the LUI project. Most of the changes that are required to add a gestural interface to a module (*e.g.*, the *videos* or *photos* apps) are located in two methods: componentDidMount() and componentWillUnmount(). In this section, we only show how we modified the *photos* application as we applied similar changes to the other modules. Furthermore, to demonstrate that adding a new component to LUI is still an easy task, we also created a PDF visualizer application.

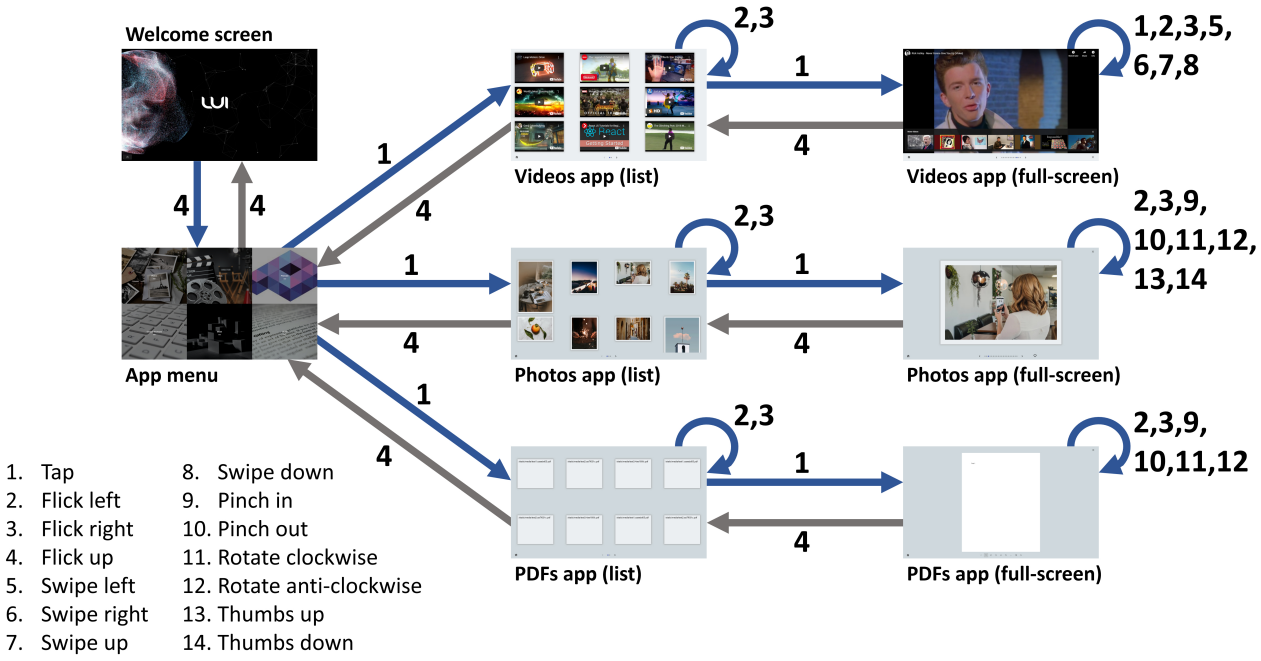


Figure 5.7: Diagram of the gestures available at any point in the LUI project.

- *Connect to the gesture recognition framework* (line 1): to connect to the framework, we simply instantiate a GestureHandler object in the componentDidMount() method. Once instantiated, it regularly attempts to establish a connection with the framework.
- *Display the position of the hand* (line 13): we use the GestureHandler's onFrame() method to update the position of the hand displayed on the screen every time the framework sends a new frame.
- *Show when a gesture is recognized* (line 2): using the onEachGesture() method, we can temporarily change the color of the fingers displayed on the screen each time any dynamic gesture is recognized.
- *React to dynamic gestures* (lines 3, 4, 5, 9): we defined *handler* functions for the four types of dynamic gestures using the onGesture() method.
- *React to static gestures* (lines 6-8 and 10-12): because static gestures are only used in full-screen mode, we dynamically add and remove *handlers* using *resp.* the onPose() and removePoseHandler() methods.

- *Disconnect from the framework* (line 14): when the application is closed, we call the disconnect() method to close the connection with the framework properly.

```
componentDidMount() {
    let gestureHandler = new GestureHandler();                          // 1
    this.gestureHandler = gestureHandler;

    gestureHandler.onEachGesture(function (gesture) {                   // 2
        this.setState({ pause: 4 });
    }.bind(this));

    gestureHandler.onGesture("rhand_lswipe", function () {              // 3
        this.props.handleSwipe("left");
    }.bind(this));

    gestureHandler.onGesture("rhand_rswipe", function () {              // 4
        this.props.handleSwipe("right");
    }.bind(this));

    gestureHandler.onGesture("rhand_uswipe", function () {              // 5
        let clicked = this.state.clicked;
        if (clicked != -1) {
            gestureHandler.removePoseHandler("grab");                   // 6
            gestureHandler.removePoseHandler("pinch");                  // 7
            gestureHandler.removePoseHandler("thumb");                  // 8
            clicked = -1;
            this.setState({ clicked });
        }
        this.props.handleSwipeUp();
    }.bind(this));

    gestureHandler.onGesture("rindex_airtap", function () {             // 9
        var { clicked, hovered } = this.state;
        if (clicked == -1 && hovered != -1) {
            gestureHandler.onPose("grab", function (data) {             // 10
                this.props.handleRotate(data.rotation);
            }.bind(this));
            gestureHandler.onPose("pinch", function (data) {            // 11
                this.props.handlePinch(data.pinch, data.translation);
            }.bind(this));
            gestureHandler.onPose("thumb", function (data) {            // 12
                let thumbUp = data.thumbVector[1] > 40;
                let thumbDown = data.thumbVector[1] < -40;
                if (thumbUp) {
                    this.props.handleThumb(true);
                }
                if (thumbDown) {
                    this.props.handleThumb(false);
                }
            }.bind(this));
            clicked = hovered;
```

```
            this.props.handleClick(clicked);
            this.setState({ clicked: clicked, hovered: -1});
        }
    }.bind(this));

    this.gestureHandler.onFrame(function (frame) {                    // 13
        if (frame.fingers.length != 0) {
            this.setState({ hand: true });
        } else {
            this.setState({ hand: false });
        }
        this.traceFingers(frame.fingers);
    }.bind(this));

    this.timer = setInterval(() => {
        if (this.state.pause > 0) {
            this.setState({ pause: this.state.pause - 1 });
        }
        if (this.state.hand) {
            var { clicked, hovered } = this.state;
            if (clicked == -1 && !this.state.amiclicked) {
                hovered = this.checkHover();
                this.setState({ hovered });
                this.props.handleHover(hovered);
            }
        }
    }, 100);
}

componentWillUnmount() {
    clearInterval(this.timer);
    this.gestureHandler.disconnect();                                 // 14
}
```

5.5 Conclusion

This chapter presented a third illustration of developing an LMC-based gesture UI for a real-world, large-scale project, *i.e.,* the LUI project. For this purpose, two preliminary gesture elicitation studies were conducted for photo and video browsing in order to explore a set of potential user-defined gestures for a series of actions on photos and videos. Based on these preliminary results, a new gesture elicitation study was conducted with an updated experimental protocol to come up with the final gesture set corresponding to the LUI actions. A benchmarking of implemented gesture recognizers on this new dataset was then conducted to determine the most appropriate gesture recognizers and to configure them for a fully running gesture-based UI. This chapter completes a series of three illustrations of how the framework can be used for developing, testing, and benchmarking LMC-based UIs.

Chapter 6

Evaluation

In Chapter 3, we developed a framework for engineering gesture-based UIs using the LMC and we subsequently illustrated it through three use cases, including a real-world, large-scale one, the LUI project (Chapter 5). Therefore, any evaluation could be conducted at two levels: (1) at the level of an LMC-based application developed using this framework by gathering input from end-users; and (2) at the level of the framework itself with input from developers and designers.

Regarding the first level, although the hypothesis H4 (Section 1.2.2) specifies that end-users' aspects are out of scope, we still want to sketch a plan for evaluating a gesture-based UI. As such, in Section 6.1, we propose a plan for evaluating the LUI-project, that can be generalized to evaluate any other gesture-based UI. Regarding the second level, based on hypothesis H3, Section 6.2 addresses the evaluation of the framework with input from developers and designers, gauges the quality of the framework, and determines whether it meets the requirements defined in Section 3.1.

6.1 At the Level of a Gesture-Based User Interface With the Framework

When designing UIs, it is crucial to always keep users in mind [87,88] through usability or user experience. As such, ensuring that they are satisfied with the gestural interface is a crucial part of the development process of any interactive application in general as well as for any gesture-based UI in particular, whether it is using an LMC or not.

Usability, as an ISO 25010 [50] software quality property, is defined as[1]: "the degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use." While *effectiveness* should evaluate how the UI supports users in completing actions accurately and *efficiency* should evaluate how end-users can perform tasks quickly through the UI, *satisfaction* should capture the subjective satisfaction with which end-users are carrying out their tasks.

We are not aware today of any method for evaluating a gesture-based UI either in general or in particular for the LMC. All that we know is a very specific adaptation of the SUS scale into a Gesture Usability Scale (GUS) [130], which added to the System Usability Scale (SUS) [13] five questions related to a gesture-based UI. These items assess the perceived reliability, performance, and compliance with the end-user's expectations, and not the usability or user experience. Later, they present a model considering accuracy, fatigue, naturalness, and gesture duration. In the area of gesture-based games, Winckler *et al.* [107] surveyed ten usability evaluations to conclude

[1] See https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&start=3

that generic questionnaires were the most used instruments (used in 70% of the evaluations), followed by interviews (30%), and observation and video recording (20%).

Rico *et al.* [96] also confirm that no such evaluation method exists today that is particularly tailored to gesture-based UI, but mention several aspects subject to evaluation, such as memorability [84], social acceptance, and usability or user experience. However, identifying the right set of properties to properly evaluate a gesture-based UI remains an open question, especially when this set is potentially large and covers system aspects as well as end-users' aspects.

Therefore, defining and validating an overall method for evaluating the usability or the user experience of gesture-based UI would represent a significant contribution that is beyond the scope of this thesis. Equally important is to assess to what extent the ecological validity of the evaluation method will be covered, which could be assessed through the ECOVAL framework [52], and the process model to be followed for applying usability engineering. Kieffer *et al.* [53] classify user experience process models into five categories. In the absence of such an evaluation method and such a process, this section sketches a plan for evaluating the user satisfaction of the LUI gesture-based UI described in Chapter 5.

6.1.1 Plan for Evaluating User Satisfaction

To evaluate user satisfaction, we create a series of scenarios. Each scenario comprises of three elements: (1) a name, which represents its goal; (2) a context; and (3) a series of steps to perform to reach this goal. The scenarios represent actions that potential users of the system are likely to perform, and they cover most of the functionalities that we have implemented. We defined eight scenarios:

1. **Open an application.**
   - Context: you are looking at the *welcome screen* and you want to open the *Videos* app.
   - Steps: (1) enter to the *app menu* by doing a "flick up" motion; (2) hover your hand above the *Videos* app; (3) "tap" to open it.
2. **View a picture.**
   - Context: you are currently looking at the home screen of the *Photos* app, you find a picture that you like and want to view it in full-screen.
   - Steps: (1) go to the second page of the gallery with a "flick left" motion; (2) hover your hand above the picture that you like; (3) "tap" to view it in full-screen.
3. **Manipulate a picture.**
   - Context: you are looking at a picture in full-screen when a detail suddenly picks your interest. You decide to manipulate the picture to see it better.
   - Steps: (1) rotate the picture by 90° clockwise with a "grab" gesture; (2) zoom in on a detail of the image using a "pinch" gesture.
4. **"Like" some pictures.**
   - Context: you are browsing through all your pictures to "like" those that you prefer.
   - Steps: (1) "like" the first with a "thumbs up" gesture; (2) go to the next picture with a "flick left" motion; (3) "like" this picture; (4) after all, you did not like the previous picture, return to this picture with a "flick right" motion; (5) remove the like with a "thumbs down" gesture.
5. **View a video.**

- Context: you are looking at the home screen of the *Videos* app and decide to watch one of the videos.
- Steps: (1) hover over the video that you like; (2) "tap" to view it in full-screen; (3) "tap" to start it.

6. **Change the volume of a video.**
   - Context: you are watching a video, but the volume is too high. You decide to decrease the volume to a more acceptable level.
   - Steps: (1) point your index; (2) decrease the volume to half its initial value by moving your hand down while keeping this pose.
7. **Go to a specific part of a video.**
   - Context: you are watching a video and you decide to go to a part of it that you enjoy.
   - Steps: (1) point your index; (2) move in the video by moving your hand to the left (while keeping this pose) until you reach the middle of the video.
8. **Return to the *app menu*.**
   - Context: you are watching a slightly boring video, and you want to go back to the *app menu*.
   - Steps: (1) pause the video with a "tap"; (2) disable full-screen with a "flick up" motion; (3) close the app with a "flick up" motion.

At the start of the experiment, participants are installed in a setup similar to the one used in our elicitation study, *i.e.,* at a table, facing a large screen that displays the application. An LMC should be placed on the table, close enough for participants' to reach it easily. Once comfortably installed, they are asked to perform each scenario in a randomized order. They should be correctly informed of the goal to reach, and the state of the application should correspond to the context of the scenario. After each scenario, regardless of whether they have reached the goal or not, they are asked to evaluate how easy the scenario was to perform on a scale from 1 (very easy) to 5 (very hard).

After they have completed every scenario, participants have to fill in a questionnaire, divided into four parts. Its first part contains a few questions that give us insight into their age, background, and prior experience with computers, smartphones, and 3D motion controllers (*e.g.,* the LMC):

- What is your age?
- What is your current occupation?
- What is your dominant hand?
- On a scale from 1 (never) to 5 (very often), how often do you use a computer?
- On a scale from 1 (never) to 5 (very often), how often do you use a smartphone/tablet?
- On a scale from 1 (never) to 5 (very often), how often do you use motion-sensing devices?

The second part asks them to evaluate the usability of the system. It contains the ten questions from John Brooke's System Usability Scale (SUS) [13]. As it is a Likert scale, answers can range between 1 (strongly disagree) and 5 (strongly agree):

- I think that I would like to use this system frequently.
- I found the system unnecessarily complex.
- I thought the system was easy to use.
- I think that I would need the support of a technical person to be able to use this system.
- I found the various functions in this system were well integrated.

- I thought there was too much inconsistency in this system.
- I would imagine that most people would learn to use this system very quickly.
- I found the system very cumbersome to use.
- I felt very confident using the system.
- I needed to learn a lot of things before I could get going with this system.

The third part of the questionnaire consists of two open-ended questions:

- If you could not perform one of the scenarios. What do you think is the reason?
- Would you change any gestures? If you answered yes, why? What kind of gesture would you propose?

Finally, we invite participants to give any feedback related to the application and the tasks they had to perform.

6.2 At the Level of the Framework Itself

This section is divided into two parts: a discussion of the framework's strengths and shortcomings based on the three selected software quality properties and an estimation of the development effort with the framework.

6.2.1 Does the Framework Meet its Requirements?

In Section 3.1, we defined the requirements that the framework should fulfill, in the light of three ISO/IEC 205010 [50] software quality properties: compatibility, maintainability, and portability. Now, we go through each one of these requirements to summarize the strengths and shortcomings of the framework. This discussion is based on the experience we gained in reviewing the related work, developing the framework, and applying it in the three use cases. Although mainly qualitative, this discussion also attempts to characterize quantitatively the extent to which each factor or sub-factor has been satisfied.

1. **Compatibility** (100%)
    - *Co-existence* (100%): because it relies on the Leap Motion API to connect to the LMC, the framework behaves like any other LMC-compatible application. As such, it does not affect other applications and is not affected by them either.
    - *Interoperability* (100%): the framework can support any skeleton-based gesture set, provided that there is an adaptation layer that can transform the gesture set into the standardized format used by the framework (see Sections 3.2.2 and 3.2.3).
2. **Maintainability** (80%)
    - *Modularity* (60%): the framework is divided into a series of interchangeable modules described in Section 3.2.3. However, because of its fixed pipeline (see 3.2.2) the current architecture is quite restrictive and it may be interesting to move towards a more flexible "block-based" architecture. In such an architecture, components would consist of separate blocks with well-defined inputs and outputs. These components could be arbitrarily combined to form a gesture-recognition pipeline.
    - *Reusability* (100%): we confirmed the reusability of the framework by using it in two applications of different scales, the DICOM viewer and the LUI project (*resp.*, Section 3.3 and Chapter 5).

- *Modifiability* (75%): we showed that adding new modules is easy by developing a series of modules for each hot spot of the framework (see Section 3.2.3). However, its architecture limits what developers could potentially achieve: it only works with skeleton data and the fixed pipeline may prevent developers from trying out more advanced recognition techniques, *e.g.,* using only one recognizer for both static and dynamic gestures.
- *Testability* (85%): we demonstrated how the framework can be used to compare gesture recognizers in Chapter 4 and, to a lesser extend, in Section 5.3. It currently only supports user-independent testing but supporting user-dependent testing should require only some small changes.

3. **Portability** (85%)

   - *Adaptability* (70%): the configuration file allows developers to configure the framework to their needs quickly. However, there is still some room for improvement. For instance, it is not yet possible to dynamically change the gesture associated with an action, and configuring the framework for an application may take some time. To speed up this process, the framework could attempt to automatically propose an adequate configuration based on the type of gestures that it should recognize.
   - *Installability* (85%): because it is written in JavaScript, the framework can be installed on most platforms. However, the installation process is too complex for less tech-savvy users. In the future, we should provide a quick and easy way to install it.
   - *Replaceability* (100%): developers can re-use existing functions from their application and call them every time a gesture is detected using the methods of the app-interface module (see Section 3.2.2).

6.2.2 Qualitative Evaluation of the Framework

To quantitatively evaluate the development effort induced by the framework, an evaluation plan could consist of a between-subject design comparing two groups of developers: one group using the framework and one group relying on the programmatic approach. An ideal evaluation would consider multiple levels of experience (*e.g.,* the nine development expertise levels defined by Costabile *et al.* [31], ranging from "customizing components" to "Multiple language and paradigm programming"), multiple levels of project complexity (*e.g.,* ranging from easy such as the DICOM Viewer to complex such as the LUI project), multiple framework configurations (*e.g.,* different gesture sets, recognizers, configuration files). Participants could then be instructed to complete the LMC-based UI with or without the framework and we could then collect quantitative measures such as, but not limited to: lines of code (LOC), modeling time, programming time, perceived complexity, or total cost of ownership. Since conducting this ideal method would require a huge amount of resources (*e.g.,* professional developers are hard to find and expensive to pay), we prefer to consider a qualitative evaluation of the framework.

Therefore, this section proposes a method for evaluating whether our framework meets the needs of application developers. It consists of collecting and analyzing data from the development of gesture-enabled applications in two situations: (1) the developers use our framework; and (2) the developers rely purely on the Leap Motion API.

At the start of their project, the developer(s) of each application should answer a short questionnaire to assess their prior experience with the LMC and gesture recognition in general. It is composed of three questions whose answers can range between 1 (strongly disagree) and 5 (strongly agree):

- Me or my team already mastered the Leap Motion API before starting this project.
- Me or my team had a good knowledge of gesture recognition techniques before starting this project.
- Me or my team had a lot of experience in app development before starting this project.

At the end of the development process, the developers that used the framework should fill in a second questionnaire, inspired by John Brooke's SUS [13]. Answers can range from 1 (strongly disagree) to 5 (strongly agree):

- I am confident that I will use the framework again if I develop another gesture-enabled application in the future.
- If I added a module to the framework (*e.g.*, a new recognizer), I found it difficult to write new modules.
- I found it easy to configure the framework to suit my needs.
- I think that it would be difficult to move to a different system for gesture recognition (*i.e.*, stop using the framework and use something else instead).
- I found that the framework met my expectations.
- I felt restricted by the framework.

Aside from filling in questionnaires, the developers of each application should time all the development phases to give us an idea of how much time can be saved by using the framework. Although it is near-impossible to provide an accurate estimation, we expect significant time-savings in three phases of software development: *design*, *implementation*, and *testing*.

Figure 6.1 compares the total effort needed for developing a gesture-based application when using our framework and when implementing gesture recognition from scratch. The *implementation* and *testing* phases should be considerably shorter as developers can rely entirely on the framework for gesture recognition. Although developers may take some time to learn how to use our framework, they do not have to discover the Leap Motion API and do not need to design a gesture recognition pipeline by themselves, which reduces the duration of the *design* phase. The extend of these time-savings will depend on how familiar developers are with the LMC and gesture recognition in general. However, they should still be significant, even for experienced developers.

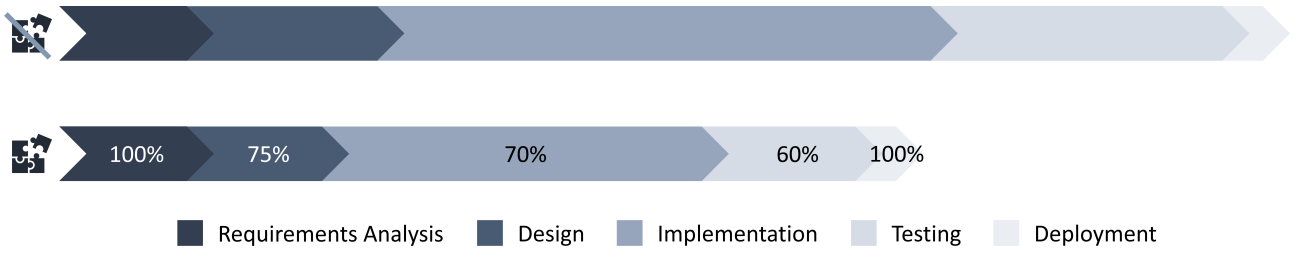


Figure 6.1: Rough qualitative estimation of the total effort needed for developing a gesture-based application with our framework (bottom) and without (top). The percentages in the bottom part give an estimation of the savings enabled by using the framework. The distribution of effort between the five development phases is based on data from [135].

Chapter 7

Final Conclusion

7.1 Overview of the Contributions

After completing this work, we are able to provide an overview of the various contributions brought throughout this manuscript.

In Chapter 1, we analyzed the context of the problem: how to effectively and efficiently develop gesture-based UIs for interactive applications using the LMC. We formulated working hypotheses and we selected three ISO 25010 software quality properties (*i.e.,* maintainability, portability, and compatibility) that were observed as unsatisfied in existing practices.

In Chapter 2, we performed a targeted literature review. We observed that the research targeting the LMC and gestural interaction in general was very prolific. However, most recognition techniques were specialized for a specific set of gestures and were not meant to be generalized to other contexts of use. On the other hand, we determined that template-based recognizers were the most suited to a generic gesture recognition framework.

Chapter 3 was dedicated to the development process of the framework. It first laid out a series of requirements and then described the design and implementation of the framework. We developed two recognizers for static gestures and six for dynamic gestures, including 3D versions of popular 2D recognizers and a novel GPSDa recognizer. We also proposed and discussed three methods for gesture segmentation. In the end, we illustrated how the framework could be used to add a gestural interface to a small application, the DICOM viewer.

In Chapter 4, we used the framework to compare the accuracy of seven recognizers on two existing datasets. We observed that $\$P^3+$ was significantly more accurate than the six other recognizers and that 16 seems to be the ideal number of training templates to maximize accuracy without impacting performance too much.

Chapter 5 described the complete process of adding a gestural interface to MIT's LUI project. We first performed an elicitation study to help us design an intuitive gesture set for the application. Then, we benchmarked a series of recognizers to maximize the recognition accuracy on this gesture set. Finally, we used this new insight to configure the framework and showed how we modified the LUI project to integrate gesture recognition. We added gestures for a total of 16 actions distributed over five modules of the LUI project, including a new PDFs application.

In Chapter 6, we proposed plans for evaluating both the utility of the framework and the user satisfaction for applications built using our framework. We also reviewed to what extend the framework met the requirements defined in Section 3.1. We found that, while it managed to meet most of our expectations, there is still a lot of room for improvement.

7.2 Future Work

The ramifications of this work are multiple. Future work could be envisioned at different levels of granularity:

- At the level of the framework software architecture: instead of relying on a pipeline architecture, the gesture recognition pipeline could be made even more modular by moving towards a "block-based" architecture, thus providing an easy way to install the framework for less tech-savvy users, or allowing end-users to submit their own gestures while the framework is executing at run-time (as opposed to at design-time).
- At the level of the framework modules:
  1. For gesture recognition by template matching: while the current software architecture is flexible enough to accommodate any new recognizer, a precise procedure should be followed for this purpose. Although this procedure is straightforward, it could become automated by offering the possibility to add, delete, modify, and configure any recognizer at run-time as opposed to design-time. In this way, a designer or any other non-expert person could submit a new recognizer to the framework without requiring any extensive programming.
  2. For gesture recognition by other machine learning methods than by template matching: while we pointed out that template matching offers the means to satisfy the quality properties, the framework could be extended with new module encapsulating machine learning techniques, in particular for benchmarking. In these conditions, these techniques should be modeled, trained, and calibrated enough to be running.
  3. For gesture segmentation: while it is essential to the proper working of gesture-enabled UIs, we barely scrapped the subject in this thesis because automated gesture segmentation is an open research problem, that is already partially addressed in some works [62]. It would be interesting to investigate methods for distinguishing intended gestures from parasitic movements in continuous streams of data, such as in [111], where Taranta *et al.* train their recognizer with automatically generated examples of parasitic gestures, a particular form of synthetic gestures [72].
- At the level of configurations: while the configuration file provides an effective way to configure the framework for a particular purpose, such as recognition, testing, or benchmarking, advanced facilities could be developed to further boost these processes by direct manipulation, by pattern matching based on existing configurations, and by learning adequate configurations from an existing base of configurations. Future work could investigate other recognition techniques. For instance, combining multiple recognizers to achieve better accuracy on a wide range of gestures, or developing more accurate recognizers which, together, would provide more accuracy.
- At the level of the LMC: the framework currently supports abstraction for one LMC, but nothing prevents it from reusing the same abstraction for exploiting many LMCs simultaneously. In this case, we are touching the field of multimodal interaction where fusion and fission of data should be parameterized in order to achieve the right configuration. For example, a two-hand interaction with one LMC for each hand represents an interesting case study. Multiple users could also be sensed by multiple LMC, thus posing the problem of user authentication and hand (right/left) identification. Computer vision techniques exist for this purpose.

Finally, while this framework only considered gestures collected using the LMC, one could focus on adding support for other types of sensors, such as smart rings and gloves, Intel RealSense cameras, or the Microsoft Kinect. But these devices require other abstractions of their raw data into higher-level data or information before handling them in a similar framework.

List of Figures

2.1 Field of view of the Leap Motion Controller (adapted from [69]). 5
2.2 Leap Motion data: palm tracking (left), fingertips (right) [70]. 6
2.3 The **SHREC2019** dataset [17]. 10
2.4 The **3DTCGS** dataset [18]. 11

3.1 Software architecture of the framework. 15
3.2 UML V2.5 Class Diagram of the Framework. 16
3.3 The DICOM application. 22
3.4 Configuration of the framework for the DICOM application. 24

4.1 Configuration of the framework for the benchmarking of recognizers. 26
4.2 Recognition rates of all recognizers for two datasets (left: SHREC2019 [17], right: 3DTCGS [18]): individual rates (top) for N=4, ..., 64 (top) and global rates (bottom) $\forall N$ ($p^{***}\leq.001$). Error bars show a confidence interval with α=.05. . . 29
4.3 Execution times of all recognizers for two datasets (left: SHREC2019 [17], right: 3DTCGS [18]): individual times for N=4, ..., 64. 32

5.1 Screenshots of the LUI application. 35
5.2 Snapshot from the GES showing a participant performing a mid-air gesture. . . 36
5.3 Aigner *et al.*'s classification of gestures [2]. 41
5.4 Articulations used in the benchmarking. 43
5.5 Confusion matrices for $\$P^3+$ and $\$P^3+X$ (N=8). 44
5.6 Configuration of the framework for the LUI application. 45
5.7 Diagram of the gestures available at any point in the LUI project. 46

6.1 Rough qualitative estimation of the total effort needed for developing a gesture-based application with our framework (bottom) and without (top). The percentages in the bottom part give an estimation of the savings enabled by using the framework. The distribution of effort between the five development phases is based on data from [135]. 54

B.1 Snapshot from the photo-browsing gesture elicitation study showing a participant performing a mid-air gesture. 79
B.2 Agreement rates for the gesture proposals (photo-browsing). Notes: referents are ordered on the horizontal axis in descending order of their agreement rates; error bars show 95% CIs computed with the AGATe tool [124]. 82
B.3 Observed percentages of gestures for each category of our taxonomy (photo-browsing). 83
B.4 Relationship between Agreement-Rate and Goodness-of-Fit for each referent (photo-browsing). 84

B.5 Relationship between Agreement-Rate and Thinking-Time (photo-browsing). 85
B.6 Agreement rates for the gesture proposals (video-browsing). Notes: referents are ordered on the horizontal axis in descending order of their agreement rates; error bars show 95% CIs computed with the AGATe tool [124]. 86
B.7 Observed percentages of gestures for each category of our taxonomy (video-browsing). 87
B.8 Relationship between Agreement-Rate and Goodness-of-Fit for each referent (video-browsing). 88
B.9 Relationship between Agreement-Rate and Thinking-Time (video-browsing). 89

C.1 Screenshots of the player and recorder tools. 93

D.1 Agreement rates for the gesture proposals. Notes: referents are ordered on the horizontal axis in descending order of their agreement rates; error bars show 95% CIs computed with the AGATe tool [124]. 95
D.2 Relationship between Agreement-Rate and Goodness-of-Fit. 96
D.3 Relationship between Agreement-Rate and Thinking-Time. 96

E.1 Illustrations of each action and its assigned gesture. 98

List of Tables

3.1 Gesture set of the DICOM application. 23

5.1 First and second most frequent gesture proposals for each referent. 39
5.2 Gesture classification according to Aigner *et al.*'s taxonomy. 41
5.3 Invariance properties of the recognizers. 42
5.4 Recognition rates of each dynamic recognizer, for T=8. 43
5.5 Recognition rates of each static recognizer, for T=64. 44

A.1 Distribution of the work between Bastien Nothomb and the author of this thesis (Arthur Sluÿters). 78

B.1 First and second most frequent gesture proposals for each referent (photo-browsing). 91
B.2 First and second most frequent gesture proposals for each referent (video-browsing). 92

E.1 Assigned gestures for each referent. 97

Glossary

ANN Artificial Neural Network.

API Application Programming Interface.

AR Augmented Reality.

CNN Convolutional Neural Network.

CNN Conditional Random Fields.

DTW Dynamic Time Warping.

FPS Frames Per Second.

FSM Finite-State Machine.

GES Gesture Elicitation Study.

HMM Hidden Markov Model.

k-NN k-Nearest Neighbor.

LOC Lines Of Code.

LUI Large User Interface.

MIT Massachusetts Institute of Technology.

RNN Recurrent Neural Network.

SDK Software Development Kit.

SVM Support Vector Machine.

TLR Target Literature Review.

UCLouvain Université Catholique de Louvain.

UI User Interface.

Bibliography

[1] Christopher Ackad, Andrew Clayphan, Martin Tomitsch, and Judy Kay. An In-the-Wild Study of Learning Mid-Air Gestures to Browse Hierarchical Information at a Large Interactive Public Display. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '15, page 1227–1238, New York, NY, USA, 2015. Association for Computing Machinery.

[2] Roland Aigner, Daniel Wigdor, Hrvoje Benko, Michael Haller, David Lindbauer, Alexandra Ion, Shengdong Zhao, and Jeffrey Tzu Kwan Valino Koh. Understanding Mid-Air Hand Gestures: A Study of Human Preferences in Usage of Gesture Types for HCI. Technical Report MSR-TR-2012-111, Microsoft Research, November 2012.

[3] Ahmad Akl and Shahrokh Valaee. Accelerometer-based gesture recognition via dynamic-time warping, affinity propagation, & compressive sensing. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2270 – 2273, 04 2010.

[4] J. Alon, V. Athitsos, Quan Yuan, and S. Sclaroff. A Unified Framework for Gesture Recognition and Spatiotemporal Gesture Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9):1685–1699, September 2009.

[5] Lisa Anthony and Jacob O. Wobbrock. A Lightweight Multistroke Recognizer for User Interface Prototypes. In *Proceedings of Graphics Interface 2010*, GI '10, page 245–252, CAN, 2010. Canadian Information Processing Society.

[6] Lisa Anthony and Jacob O. Wobbrock. $N-Protractor: A Fast and Accurate Multistroke Recognizer. In *Proceedings of Graphics Interface 2012*, GI '12, page 117–120, CAN, 2012. Canadian Information Processing Society.

[7] Daniel Ashbrook and Thad Starner. MAGIC: A Motion Gesture Design Tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 2159–2168, New York, NY, USA, 2010. Association for Computing Machinery.

[8] Danilo Avola, Marco Bernardi, Luigi Cinque, Gian Luca Foresti, and Cristiano Massaroni. Exploiting Recurrent Neural Networks and Leap Motion Controller for the Recognition of Sign Language and Semaphoric Hand Gestures. *IEEE Transactions on Multimedia*, 21(1):234–245, January 2019.

[9] M. K. Bhuyan, D. Ajay Kumar, Karl F. MacDorman, and Yuji Iwahori. A novel set of features for continuous hand gesture recognition. *Journal on Multimodal User Interfaces*, 8(4):333–343, July 2014.

[10] Mohamed-Ikbel Boulabiar, Gilles Coppin, and Franck Poirier. The Study of the Full Cycle of Gesture Interaction, The Continuum between 2D and 3D. In Masaaki Kurosu, editor,

Human-Computer Interaction. Advanced Interaction Modalities and Techniques, pages 24–35, Cham, 2014. Springer International Publishing.

[11] Said Yacine Boulahia, Eric Anquetil, Richard Kulpa, and Franck Multon. 3D Multistroke Mapping (3DMM): Transfer of Hand-Drawn Pattern Representation for Skeleton-Based Gesture Recognition. In *Proceedings of the 12th IEEE International Conference on Automatic Face Gesture Recognition*, FG '17, pages 462–467. IEEE, 2017.

[12] Andrew Bragdon, Rob DeLine, Ken Hinckley, and Meredith Ringel Morris. Code Space: Touch + Air Gesture Hybrid Interactions for Supporting Developer Meetings. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '11, page 212–221, New York, NY, USA, 2011. Association for Computing Machinery.

[13] John Brooke. SUS-A quick and dirty usability scale. *Usability evaluation in industry*, pages 189–194, 1996.

[14] R. Brunelli. *Template Matching Techniques in Computer Vision: Theory and Practice*. John Wiley & Sons, New York, 2009.

[15] Ashley Bush and Sandeep Purao. Mapping uml techniques to design activities. In Matti Rossi and Keng Siau, editors, *Information Modeling in the New Millennium*, chapter 12, pages 199–217. IDEA Publishing Group, 2011.

[16] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A Unifying Reference Framework for multi-target user interfaces. *Interact. Comput.*, 15(3):289–308, 2003.

[17] F. M. Caputo, S. Burato, G. Pavan, T. Voillemin, H. Wannous, J. P. Vandeborre, M. Maghoumi, E. M. Taranta II, A. Razmjoo, J. J. LaViola Jr., F. Manganaro, S. Pini, G. Borghi, R. Vezzani, R. Cucchiara, H. Nguyen, M. T. Tran, and A. Giachetti. Online Gesture Recognition. In Silvia Biasotti, Guillaume Lavoué, and Remco Veltkamp, editors, *Eurographics Workshop on 3D Object Retrieval*. The Eurographics Association, 2019.

[18] F. M. Caputo, P. Prebianca, A. Carcangiu, L. D. Spano, and A. Giachetti. A 3 Cent Recognizer: Simple and Effective Retrieval and Classification of Mid-Air Gestures from Single 3D Traces. In *Proceedings of the Conference on Smart Tools and Applications in Computer Graphics*, STAG '17, page 9–15, Goslar, DEU, 2017. Eurographics Association.

[19] Fabio M. Caputo, Pietro Prebianca, Alessandro Carcangiu, Lucio D. Spano, and Andrea Giachetti. Comparing 3D trajectories for simple mid-air gesture recognition. *Computers & Graphics*, 73:17 – 25, 2018.

[20] Alessandro Carcangiu and Lucio Davide Spano. G-Gene: A Gene Alignment Method for Online Partial Stroke Gestures Recognition. *Proc. ACM Hum.-Comput. Interact.*, 2(EICS), June 2018.

[21] Alessandro Carcangiu and Lucio Davide Spano. Integrating Declarative Models and HMMs for Online Gesture Recognition. In *Proceedings of the 24th International Conference on Intelligent User Interfaces: Companion*, IUI '19, page 87–88, New York, NY, USA, 2019. Association for Computing Machinery.

[22] Alessandro Carcangiu, Lucio Davide Spano, Giorgio Fumera, and Fabio Roli. Gesture Modelling and Recognition by Integrating Declarative Models and Pattern Recognition Algorithms. In Sebastiano Battiato, Giovanni Gallo, Raimondo Schettini, and Filippo Stanco, editors, *Image Analysis and Processing - ICIAP 2017*, pages 84–95, Cham, 2017. Springer International Publishing.

[23] Alessandro Carcangiu, Lucio Davide Spano, Giorgio Fumera, and Fabio Roli. DEICTIC: A compositional and declarative gesture description based on hidden markov models. *International Journal of Human-Computer Studies*, 122:113 – 132, 2019.

[24] Sait Celebi, Ali S. Aydin, Talha T. Temiz, and Tarik Arici. Gesture Recognition using Skeleton Data with Weighted Dynamic Time Warping. In *Proceedings of the International Conference on Computer Vision Theory and Applications*. SciTePress - Science and and Technology Publications, 2013.

[25] Lijiang Chen, Xingyu wu, Yuli Xue, and Xia Mao. Trajectory-based view-invariant hand gesture recognition by fusing shape and orientation. *IET Computer Vision*, 9:797–805(8), December 2015.

[26] Xiang "Anthony" Chen, Julia Schwarz, Chris Harrison, Jennifer Mankoff, and Scott E. Hudson. Air+touch: Interweaving Touch & in-Air Gestures. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, page 519–525, New York, NY, USA, 2014. Association for Computing Machinery.

[27] Hong Cheng, Lu Yang, and Zicheng Liu. Survey on 3D Hand Gesture Recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(9):1659–1673, 2016.

[28] Eunjung Choi, Heejin Kim, and Min Chung. A taxonomy and notation method for three-dimensional hand gestures. *International Journal of Industrial Ergonomics*, 44:171–188, 01 2014.

[29] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge, 2 edition, 1988.

[30] Jacob Cohen. A power primer. *Psychological Bulletin*, 112(1):155–159, 1992.

[31] Maria Francesca Costabile, Piero Mussio, Loredana Parasiliti Provenza, and Antonio Piccinno. End users as unwitting software developers. In *Proceedings of the 4th International Workshop on End-user Software Engineering*, WEUSE '08, pages 6–10, New York, NY, USA, 2008. ACM.

[32] Elena-Gina Craciun, Ionela Rusu, and Radu-Daniel Vatavu. Free-Hand Gesture Recognizer Pseudocode. http://www.eed.usv.ro/mintviz/projects/GIVISIMP/data/Pseudocode2.pdf, 2016. [Online; accessed 09-August-2020].

[33] Q. De Smedt, H. Wannous, J.-P. Vandeborre, J. Guerry, B. Le Saux, and D. Filliat. 3D Hand Gesture Recognition Using a Depth and Skeletal Dataset: SHREC'17 Track. In *Proceedings of the Workshop on 3D Object Retrieval*, 3Dor '17, page 33–38, Goslar, DEU, 2017. Eurographics Association.

[34] William Delamare, Thomas Janssoone, Céline Coutrix, and Laurence Nigay. Designing 3D Gesture Guidance: Visual Feedback and Feedforward Design Options. In *Proceedings*

of the International Working Conference on Advanced Visual Interfaces, AVI '16, page 152–159, New York, NY, USA, 2016. Association for Computing Machinery.

[35] Stefano Dessì and Lucio Davide Spano. DG3: Exploiting Gesture Declarative Models for Sample Generation and Online Recognition. *Proc. ACM Hum.-Comput. Interact.*, 4(EICS), June 2020.

[36] Giannis Drossis, Dimitris Grammenos, Chryssi Birliraki, and Constantine Stephanidis. MAGIC: Developing a Multimedia Gallery Supporting mid-Air Gesture-Based Interaction and Control. In Constantine Stephanidis, editor, *HCI International 2013 - Posters' Extended Abstracts*, pages 303–307, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[37] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley & Sons, New York, 2000.

[38] Catherine Dumas, Sanda Erdelez, Jeffrey Pomerantz, and Vik Parthiban. Usability testing of LUI: A new human-computer interface for large displays. *Proceedings of the Association for Information Science and Technology*, 56(1):645–647, 2019.

[39] Sergio Escalera, Isabelle Guyon, and Vassilis Athitsos. *Gesture Recognition*. The Springer Series on Challenges in Machine Learning. Springer International Publishing, Berlin, 1 edition, 2017.

[40] Julián García, José Pascual Molina, Diego Martínez, Arturo S. García, Pascual González, and Jean Vanderdonckt. Prototyping and evaluating glove-based multimodal interfaces. *J. Multimodal User Interfaces*, 2(1):43–52, 2008.

[41] M. Geetha and M. R. Kaimal. A 3D stroke based representation of sign language signs using key maximum curvature points and 3D chain codes. *Multim. Tools Appl.*, 77(6):7097–7130, 2018.

[42] Bogdan-Florin Gheran, Jean Vanderdonckt, and Radu-Daniel Vatavu. Gestures for Smart Rings: Empirical Results, Insights, and Design Implications. In *Proceedings of the 2018 Designing Interactive Systems Conference*, DIS '18, pages 623–635, New York, NY, USA, 2018. ACM.

[43] Daniela Grijincu, Miguel A. Nacenta, and Per Ola Kristensson. User-Defined Interface Gestures: Dataset and Analysis. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, ITS '14, page 25–34, New York, NY, USA, 2014. Association for Computing Machinery.

[44] Tibor Guzsvinecz, Veronika Szucs, and Cecilia Sik-Lanyi. Suitability of the Kinect Sensor and Leap Motion Controller—A Literature Review. *Sensors*, 19(5), 2019.

[45] James Herold and Thomas F. Stahovich. The One Cent Recognizer: A Fast, Accurate, and Easy-to-Implement Handwritten Gesture Recognition Technique. In Karan Singh and Levent Burak Kara, editors, *Eurographics Workshop on Sketch-Based Interfaces and Modeling*. The Eurographics Association, 2012.

[46] M. Hoffman, P. Varcholik, and J. J. LaViola. Breaking the status quo: Improving 3D gesture recognition with spatially convenient input devices. In *2010 IEEE Virtual Reality Conference (VR)*, pages 59–66, 2010.

[47] Christian Holz and Andrew Wilson. Data Miming: Inferring Spatial Object Descriptions from Human Gesture. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, page 811–820, New York, NY, USA, 2011. Association for Computing Machinery.

[48] Jinmiao Huang, Prakhar Jaiswal, and Rahul Rai. Gesture-based system for next generation natural and intuitive interfaces. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 33(1):54–68, 2019.

[49] IEEE. Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990.

[50] ISO. ISO/IEC 25010 - Software Quality Product Standard. standard, International Standard Organization, Geneva, 2019.

[51] M.G. Kendall and B. Babington Smith. The Problem of m Rankings. *Annals of Math. Statistics*, 10(3):275–287, 1939.

[52] Suzanne Kieffer. ECOVAL: ecological validity of cues and representative design in user experience evaluations. *AIS Trans. Hum. Comput. Interact.*, 9(2):4, 2017.

[53] Suzanne Kieffer, Luka Rukonic, Vincent Kervyn de Meerendré, and Jean Vanderdonckt. A process reference model for UX. In Ana Paula Cláudio, Kadi Bouatouch, Manuela Chessa, Alexis Paljic, Andreas Kerren, Christophe Hurter, Alain Trémeau, and Giovanni Maria Farinella, editors, *Computer Vision, Imaging and Computer Graphics Theory and Applications - 14th International Joint Conference, VISIGRAPP 2019, Prague, Czech Republic, February 25-27, 2019, Revised Selected Papers*, volume 1182 of *Communications in Computer and Information Science*, pages 128–152. Springer, 2019.

[54] Barbara Kitchenham, Rialette Pretorius, David Budgen, O. Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen Linkman. Systematic literature reviews in software engineering – A tertiary study. *Information and Software Technology*, 52(8):792–805, 2010.

[55] D. Kohlsdorf, T. Starner, and D. Ashbrook. MAGIC 2.0: A web tool for false positive prediction and prevention for gesture recognition systems. In *Face and Gesture 2011*, pages 1–6, 2011.

[56] Amit Konar and Sriparna Saha. *Gesture recognition : principles, techniques and applications*, volume 724 of *Studies in computational intelligence*. Springer, Berlin, 1 edition, 2018.

[57] M. Korkman, U. Kirk, and S. Kemp. *NEPSY: A Developmental Neuropsychological Assessment. San Antonio TX: Psychological Corporation.* Pearson Assessment, 1998.

[58] Panayiotis Koutsabasis and Panagiotis Vogiatzidakis. Empirical Research in Mid-Air Interaction: A Systematic Review. *International Journal of Human–Computer Interaction*, 35(18):1747–1768, 2019.

[59] Sven Kratz and Maribeth Back. Towards Accurate Automatic Segmentation of IMU-Tracked Motion Gestures. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '15, page 1337–1342, New York, NY, USA, 2015. Association for Computing Machinery.

[60] Sven Kratz and Michael Rohs. A $3 Gesture Recognizer: Simple Gesture Recognition for Devices Equipped with 3D Acceleration Sensors. In *Proceedings of the 15th International Conference on Intelligent User Interfaces*, IUI '10, page 341–344, New York, NY, USA, 2010. Association for Computing Machinery.

[61] Sven Kratz and Michael Rohs. Protractor3D: A Closed-Form Solution to Rotation-Invariant 3D Gestures. In *Proceedings of the 16th International Conference on Intelligent User Interfaces*, IUI '11, page 371–374, New York, NY, USA, 2011. Association for Computing Machinery.

[62] Sven Kratz and Jason Wiese. GestureSeg: Developing a Gesture Segmentation System Using Gesture Execution Phase Labeling by Crowd Workers. In *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '16, page 61–72, New York, NY, USA, 2016. Association for Computing Machinery.

[63] Per Ola Kristensson and Leif C. Denby. Continuous Recognition and Visualization of Pen Strokes and Touch-Screen Gestures. In Tracy Hammond and Andrew Nealen, editors, *Sketch Based Interfaces and Modeling, Vancouver, BC, Canada, 5-7 August 2011. Proceedings*, pages 95–102. Eurographics Association, 2011.

[64] Per-Ola Kristensson and Shumin Zhai. SHARK2: A Large Vocabulary Shorthand Writing System for Pen-based Computers. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, UIST '04, pages 43–52, New York, NY, USA, 2004. ACM.

[65] Eyal Krupka, Kfir Karmon, Noam Bloom, Daniel Freedman, Ilya Gurvich, Aviv Hurvitz, Ido Leichter, Yoni Smolin, Yuval Tzairi, Alon Vinnikov, and Aharon Bar Hillel. Toward Realistic Hands Gesture Interface: Keeping it Simple for Developers and Machines. In *CHI'17 Proceedings of the 35th Annual ACM Conference on Human Factors in Computing Systems*. ACM, April 2017.

[66] Gordon Kurtenbach, George Fitzmaurice, Thomas Baudel, and Bill Buxton. The Design of a GUI Paradigm Based on Tablets, Two-hands, and Transparency. In *Proceedings of the ACM International Conference on Human Factors in Computing Systems*, CHI '97, pages 35–42, New York, NY, USA, 1997. ACM.

[67] Lynn Kysh. Difference between a systematic review and a literature review, 8 2013.

[68] Joseph J. LaViola. 3D Gestural Interaction: The State of the Field. *ISRN Artificial Intelligence*, 2013:514641, December 2013. Publisher: Hindawi Publishing Corporation.

[69] Inc. Leap Motion. Leap Motion - Horizontal View Angle. https://di4564baj7skl.cloudfront.net/documentation/images/Leap_horizontalViewAngle.png, 2020. [Online; accessed 10-August-2020].

[70] Inc. Leap Motion. Leap Motion API Overview. https://developer-archive.leapmotion.com/documentation/csharp/devguide/Leap_Overview.html, 2020. [Online; accessed 29-July-2020].

[71] Luis A. Leiva, Daniel Martín-Albo, and Réjean Plamondon. Gestures à Go Go: Authoring Synthetic Human-Like Stroke Gestures Using the Kinematic Theory of Rapid Movements. *ACM Transactions on Intelligent Systems Technology*, 7(2), nov 2015.

[72] Luis A. Leiva, Daniel Martín-Albo, and Radu-Daniel Vatavu. GATO: Predicting Human Performance with Multistroke and Multitouch Gesture Input. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '18, New York, NY, USA, 2018. Association for Computing Machinery.

[73] Yang Li. Protractor: A Fast and Accurate Gesture Recognizer. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2169–2172, New York, NY, USA, 2010. ACM.

[74] Allan Christian Long, James A. Landay, and Lawrence A. Rowe. Implications for a Gesture Design Tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, page 40–47, New York, NY, USA, 1999. Association for Computing Machinery.

[75] Tahir Mustafa Madani, Muhammad Tahir, Sheikh Ziauddin, Syed Raza, and Mirza Ahmed. An accelerometer-based approach to evaluate 3D unistroke gestures. *The International Arab Journal of Information Technology*, 12(4):389–394, 2015.

[76] Naveen Madapana, Glebys Gonzalez, Rahul Taneja, Richard Rodgers, Lingsong Zhang, and Juan Wachs. Preference Elicitation: Obtaining Gestural Guidelines for PACS in Neurosurgery. *International Journal of Medical Informatics*, 130, 07 2019.

[77] Mehran Maghoumi and Joseph J. LaViola. DeepGRU: Deep Gesture Recognition Utility. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Daniela Ushizima, Sek Chai, Shinjiro Sueda, Xin Lin, Aidong Lu, Daniel Thalmann, Chaoli Wang, and Panpan Xu, editors, *Advances in Visual Computing*, pages 16–31, Cham, 2019. Springer International Publishing.

[78] Nathan Magrofuoco, Jorge Luis Pérez-Medina, Paolo Roselli, Jean Vanderdonckt, and Santiago Villarreal. Eliciting Contact-Based and Contactless Gestures With Radar-Based Sensors. *IEEE Access*, 7:176982–176997, 2019.

[79] Mentar Mahmudi, Pawan Harish, Benoît [Le Callennec], and Ronan Boulic. Artist-oriented 3D character posing from 2D strokes. *Computers & Graphics*, 57:81 – 91, 2016.

[80] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. Hand gesture recognition with Leap Motion and Kinect devices. *2014 IEEE International Conference on Image Processing, ICIP 2014*, pages 1565–1569, 2014.

[81] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. Hand Gesture Recognition with Jointly Calibrated Leap Motion and Depth Sensor. *Multimedia Tools Appl.*, 75(22):14991–15015, November 2016.

[82] Antigoni Mezari and Ilias Maglogiannis. Gesture Recognition Using Symbolic Aggregate Approximation and Dynamic Time Warping on Motion Data. In *Proceedings of the 11th EAI International Conference on Pervasive Computing Technologies for Healthcare*, PervasiveHealth '17, page 342–347, New York, NY, USA, 2017. Association for Computing Machinery.

[83] Meredith Ringel Morris, Andreea Danielescu, Steven Drucker, Danyel Fisher, Bongshin Lee, m. c. schraefel, and Jacob O. Wobbrock. Reducing Legacy Bias in Gesture Elicitation Studies. *Interactions*, 21(3):40–45, May 2014.

[84] Miguel A. Nacenta, Yemliha Kamber, Yizhou Qiang, and Per Ola Kristensson. Memorability of Pre-Designed and User-Defined Gesture Sets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 1099–1108, New York, NY, USA, 2013. Association for Computing Machinery.

[85] Abhishek Nandy. *Leap Motion for Developers*. Apress Media, LLC, New York, NY, USA, 2016.

[86] Pedro Neto, Dário Pereira, J. Norberto Pires, and A. Moreira. Real-time and continuous hand gesture spotting: An approach based on artificial neural networks. In *2013 IEEE International Conference on Robotics and Automation*, pages 178–183, 2013.

[87] J. Nielsen. *Usability Engineering*. Interactive Technologies. Elsevier Science, 1994.

[88] Jakob Nielsen. Response Times: The 3 Important Limits. https://www.nngroup.com/articles/response-times-3-important-limits/, 1993. [Online; accessed 20-July-2020].

[89] Vik Parthiban and Ashley Jieun Lee. LUI: A Multimodal, Intelligent Interface for Large Displays. In *The 17th International Conference on Virtual-Reality Continuum and Its Applications in Industry*, VRCAI '19, New York, NY, USA, 2019. Association for Computing Machinery.

[90] Julien Pauchot, Laetitia Di Tommaso, Ahmed Lounis, Mourad Benassarou, Pierre Mathieu, Dominique Bernot, and Sébastien Aubry. Leap Motion Gesture Control With Carestream Software in the Operating Room to Control Imaging: Installation Guide and Discussion. *Surgical Innovation*, 22(6):615–620, 2015. PMID: 26002115.

[91] Thammathip Piumsomboon, Adrian Clark, Mark Billinghurst, and Andy Cockburn. User-defined Gestures for Augmented Reality. In *Proc. of INTERACT '13*, pages 282–299. Springer Berlin Heidelberg, 2013.

[92] Wolfgang Pree. Meta Patterns - A Means For Capturing the Essentials of Reusable Object-Oriented Design. In *Proceedings of the 8th European Conference on Object-Oriented Programming*, ECOOP '94, page 150–162, Berlin, Heidelberg, 1994. Springer-Verlag.

[93] Siddharth S. Rautaray and Anupam Agrawal. Vision based hand gesture recognition for human computer interaction: a survey. *Artif. Intell. Rev.*, 43(1):1–54, 2015.

[94] J. Reaver, T. F. Stahovich, and J. Herold. How to Make a Quick$: Using Hierarchical Clustering to Improve the Efficiency of the Dollar Recognizer. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, pages 103–108, New York, NY, USA, 2011. ACM.

[95] Microsoft Research. Project Gesture. https://docs.microsoft.com/en-us/gestures/, 2017. [Online; accessed 23-July-2020].

[96] Julie Rico, Andrew Crossan, and Stephen A. Brewster. Gesture-based interfaces: Practical applications of gestures in real world mobile settings. In David England, editor, *Whole Body Interaction*, Human-Computer Interaction Series, pages 173–186. Springer, 2011.

[97] Dean Rubine. Specifying Gestures by Example. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, pages 329–337, New York, NY, USA, 1991. ACM.

[98] Ruifeng Yuan, Jun Cheng, Pengcheng Li, Guang Chen, Can Xie, and Qi Xie. View invariant hand gesture recognition using 3D trajectory. In *2010 8th World Congress on Intelligent Control and Automation*, pages 6315–6320, 2010.

[99] Jaime Ruiz, Yang Li, and Edward Lank. User-defined Motion Gestures for Mobile Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 197–206, New York, NY, USA, 2011. ACM.

[100] Priscila T. M. Saito, Rodrigo Y. M. Nakamura, Willian P. Amorim, João P. Papa, Pedro J. de Rezende, and Alexandre X. Falcão. Choosing the Most Effective Pattern Classification Model under Learning-Time Constraint. *PLOS ONE*, 10(6):1–23, 06 2015.

[101] Brandon Sanders. *Mastering Leap Motion*. Packt Publishing, New York, NY, USA, 11 2014.

[102] Shlomo S. Sawilowsky. A Different Future For Social And Behavioral Science Research. *Journal of Modern Applied Statistical Method*, 2(1):128–132, 2003.

[103] V. M. Sethu Janaki, S. Babu, and S. S. Sreekanth. Real time recognition of 3D gestures in mobile devices. In *2013 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, pages 149–152, 2013.

[104] Jia Sheng. A Study of AdaBoost in 3D Gesture Recognition. technical report CSC2515, Department of Computer Science, University of Toronto, 2004.

[105] Beat Signer, Ueli Kurmann, and Moira C. Norrie. iGesture: A General Gesture Recognition Framework. In *9th International Conference on Document Analysis and Recognition (ICDAR 2007), 23-26 September, Curitiba, Paraná, Brazil*, pages 954–958. IEEE Computer Society, 2007.

[106] Beat Signer, Moira C. Norrie, and Ueli Kurmann. iGesture. A Java framework for the development and deployment of stroke-based online Gesture recognition algorithms. Technical report, ETH Zurich, Zürich, 2011. Technical Reports D-INFK.

[107] Fernando Winckler Simor, Manoela Rogofski Brum, Jaison Dairon Ebertz Schmidt, Rafael Rieder, and Ana Carolina Bertoletti De Marchi. Usability Evaluation Methods for Gesture-Based Games: A Systematic Review. *JMIR Serious Games*, 4(2):e17, 10 2016.

[108] Mischa Spiegelmock. *Leap Motion Development Essentials*. Packt Publishing, 1 edition, October 2013.

[109] Muhammad Tahir, Tahir Mustafa Madani, Sheikh Ziauddin, Muhammad Arshad Awan, and Waqar Hussain Rana. Wiimote squash: comparing DTW and WFM techniques for 3D gesture recognition. *Int. Arab J. Inf. Technol.*, 11(4):362–369, 2014.

[110] Eugene M. Taranta, II and Joseph J. LaViola, Jr. Penny Pincher: A Blazing Fast, Highly Accurate $-family Recognizer. In *Proceedings of the 41st Graphics Interface Conference*, GI '15, pages 195–202, Toronto, Ont., Canada, Canada, 2015. Canadian Information Processing Society.

[111] Eugene M. Taranta II, Amirreza Samiei, Mehran Maghoumi, Pooya Khaloo, Corey R. Pittman, and Joseph J. LaViola Jr. Jackknife: A Reliable Recognizer with Few Samples and Many Modalities. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 5850–5861. ACM, 2017.

[112] Harry Usuwiel, Jean Wilberz, Tancrède d'Aspremont Lynden, and Bastien Nothomb. Opportunity study for the implementation of gesture recognition technology in the hospital environment Commercialization through a university spin-off. Master's thesis, Université catholique de Louvain, 2020.

[113] Nicanor Valdez, Ronnie Besas, China Yu, Donna Dumalaon, and Rowel Atienza. 3D gestures on 2D screens for mobile games. In *2014 IEEE Asia Pacific Conference on Wireless and Mobile*, pages 232–237, 2014.

[114] Jean Vanderdonckt, Nathan Magrofuoco, Suzanne Kieffer, Jorge Pérez, Ysabelle Rase, Paolo Roselli, and Santiago Villarreal. Head and Shoulders Gestures: Exploring User-Defined Gestures with Upper Body. In Aaron Marcus and Wentao Wang, editors, *Design, User Experience, and Usability. User Experience in Advanced Technological Environments*, pages 192–213, Cham, 2019. Springer International Publishing.

[115] Jean Vanderdonckt, Paolo Roselli, and Jorge Luis Pérez-Medina. !FTL, an Articulation-Invariant Stroke Gesture Recognizer with Controllable Position, Scale, and Rotation Invariances. In *Proceedings of the 20th ACM International Conference on Multimodal Interaction*, ICMI '18, page 125–134, New York, NY, USA, 2018. Association for Computing Machinery.

[116] Radu-Daniel Vatavu. The Effect of Sampling Rate on the Performance of Template-Based Gesture Recognizers. In *Proceedings of the 13th International Conference on Multimodal Interfaces*, ICMI '11, page 271–278, New York, NY, USA, 2011. Association for Computing Machinery.

[117] Radu-Daniel Vatavu. 1F: One Accessory Feature Design for Gesture Recognizers. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces*, IUI '12, pages 297–300, New York, NY, USA, 2012. ACM.

[118] Radu-Daniel Vatavu. The Impact of Motion Dimensionality and Bit Cardinality on the Design of 3D Gesture Recognizers. *Int. J. Hum.-Comput. Stud.*, 71(4):387–409, April 2013.

[119] Radu-Daniel Vatavu. Improving Gesture Recognition Accuracy on Touch Screens for Users with Low Vision. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 4667–4679, New York, NY, USA, 2017. Association for Computing Machinery.

[120] Radu-Daniel Vatavu. The Dissimilarity-Consensus Approach to Agreement Analysis in Gesture Elicitation Studies. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, New York, NY, USA, 2019. Association for Computing Machinery.

[121] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. Gestures as Point Clouds: A $P Recognizer for User Interface Prototypes. In *Proceedings of the 14th ACM International*

Conference on Multimodal Interaction, ICMI '12, page 273–280, New York, NY, USA, 2012. Association for Computing Machinery.

[122] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. $Q: A Super-Quick, Articulation-Invariant Stroke-Gesture Recognizer for Low-Resource Devices. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '18, New York, NY, USA, 2018. Association for Computing Machinery.

[123] Radu-Daniel Vatavu and Ştefan-Gheorghe Pentiuc. Multi-Level Representation of Gesture as Command for Human-Computer Interaction. *Computing and Informatics*, 27(6):837–851, 2008.

[124] Radu-Daniel Vatavu and Jacob O. Wobbrock. Formalizing Agreement Analysis for Elicitation Studies: New Measures, Significance Test, and Toolkit. In *Proc. of the 33rd ACM Conf. on Human Factors in Computing Systems*, CHI '15, pages 1325–1334, New York, NY, USA, 2015. ACM.

[125] Radu-Daniel Vatavu and Ionut-Alexandru Zaiti. Leap Gestures for TV: Insights from an Elicitation Study. In *Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video*, TVX '14, page 131–138, New York, NY, USA, 2014. Association for Computing Machinery.

[126] Santiago Villarreal-Narvaez, Jean Vanderdonckt, Radu-Daniel Vatavu, and Jacob O. Wobbrock. A systematic review of gesture elicitation studies: What can we learn from 216 studies? In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, DIS '20, page 855–872, New York, NY, USA, 2020. Association for Computing Machinery.

[127] Q. Wang, Y. Xu, X. Bai, D. Xu, Y. Chen, and X. Wu. Dynamic gesture recognition using 3D trajectory. In *2014 4th IEEE International Conference on Information Science and Technology*, pages 598–601, 2014.

[128] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. Analysis of the Accuracy and Robustness of the Leap Motion Controller. *Sensors (Basel, Switzerland)*, 13:6380–6393, 05 2013.

[129] Tracy Westeyn, Helene Brashear, Amin Atrash, and Thad Starner. Georgia Tech Gesture Toolkit: Supporting Experiments in Gesture Recognition. In *Proceedings of the 5th International Conference on Multimodal Interfaces*, ICMI '03, page 85–92, New York, NY, USA, 2003. Association for Computing Machinery.

[130] Daniel Wickeroth, Paul Benölken, and Ulrich Lang. Manipulating 3D Content using Gestures in Design Review Scenarios. *International Journal of Information Studies*, 1(4):242–250, 10 2009.

[131] Jacob O. Wobbrock, Htet Htet Aung, Brandon Rothrock, and Brad A. Myers. Maximizing the Guessability of Symbolic Input. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '05, pages 1869–1872, New York, NY, USA, 2005. ACM.

[132] Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. User-defined gestures for surface computing. In *Proc. of the CHI Conf. on Human Factors in Computing Systems*, CHI '09, pages 1083–1092, New York, NY, USA, 2009. ACM.

[133] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, page 159–168, New York, NY, USA, 2007. Association for Computing Machinery.

[134] Carl D. Worth. xstroke: Full-screen Gesture Recognition for X. In *Proceedings of the FREENIX Track: 2003 USENIX Annual Technical Conference, June 9-14, 2003, San Antonio, Texas, USA*, pages 187–196, 2003.

[135] Ye Yang, Mei He, Mingshu Li, Qing Wang, and Barry Boehm. Phase Distribution of Software Development Effort. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, page 61–69, New York, NY, USA, 2008. Association for Computing Machinery.

[136] Mais Yasen and Shaidah Jusoh. A systematic review on hand gesture recognition techniques, challenges and applications. *PeerJ Computer Science*, 5:e218, September 2019.

[137] Ionuţ-Alexandru Zaiţi, Ştefan Gheorghe Pentiuc, and Radu-Daniel Vatavu. On free-hand tv control: Experimental results on user-elicited gestures with leap motion. *Pers. Ubiquit. Comput.*, 19(5–6):821–838, 2015.

[138] Shumin Zhai, Per Ola Kristensson, Caroline Appert, Tue Haste Anderson, and Xiang Cao. Foundational Issues in Touch-Surface Stroke Gesture Design — An Integrative Review. *Foundations and Trends in Human–Computer Interaction*, 5(2):97–205, 2012.

Appendix A

Work Distribution

Part of the work presented in this thesis was performed in collaboration with Bastien Nothomb [112]. Table A.1 details how the work was distributed between him and the author of this thesis (Arthur Sluÿters).

Framework Architecture	
V1 (June 2020)	
Gesture recognition pipeline	Bastien Nothomb & Arthur Sluÿters
Gesture set representation (gestures)	Bastien Nothomb
Unified format for storing datasets	Bastien Nothomb
Configuration file	Arthur Sluÿters
App-interface	Arthur Sluÿters
V2 (August 2020)	
Support for static gesture sets	Arthur Sluÿters
Dataset manipulation (configuration file)	Arthur Sluÿters
Code refactoring	Arthur Sluÿters
Modules	
Sensors	
Leap-sensor	Bastien Nothomb & Arthur Sluÿters
Datasets	
LUI-dynamic	Bastien Nothomb & Arthur Sluÿters
LUI-static	Arthur Sluÿters
Classifiers	
$P^3+	Arthur Sluÿters
GPSDa	Arthur Sluÿters
Analyzers	
Basic-analyzer	Arthur Sluÿters
Segmenters	
Left-hand segmenter	Arthur Sluÿters
Zoning segmenter	Bastien Nothomb
Window segmenter	Arthur Sluÿters
Recognizers	
3 cent	Arthur Sluÿters
$P^3	Arthur Sluÿters
$P^3+	Arthur Sluÿters
$P^3+X	Arthur Sluÿters
$Q^3	Arthur Sluÿters
Jackknife	Arthur Sluÿters
Benchmarking Tool	
V1 (June 2020)	
Benchmarking pipeline	Bastien Nothomb
Configuration file	Bastien Nothomb
V2 (August 2020)	
Output results in JSON format	Arthur Sluÿters
Support for static gesture recognizers	Arthur Sluÿters
Code refactoring	Arthur Sluÿters
Applications	
DICOM viewer	Bastien Nothomb
LUI project (gestural interface)	Arthur Sluÿters

Table A.1: Distribution of the work between Bastien Nothomb and the author of this thesis (Arthur Sluÿters).

Appendix B

Preliminary Gesture Elicitation Studies

B.1 General Information About the Experiments

Two preliminary gesture elicitation studies [126,131,132] following the classical methodology [42,132] to collect users' preferences for mid-air gesture input. This section will provide general information about these experiments, such as how they were designed and what measures were employed.

B.1.1 Participants

Participants were selected to get a representative sample of potential users of large UIs.

B.1.2 Apparatus

We did not have a Kinect or Leap Motion device at our disposal for the experiment. However, due to the "non-wearable" nature of such motion-sensing devices, having such a device was not strictly required to complete the experiment (as opposed to wearable input devices, which can be needed by the participants to perform specific gestures on their surfaces). Thus, participants were asked to execute gestures in mid-air in front of a screen, as if it was able to distinguish any of their hand gestures.

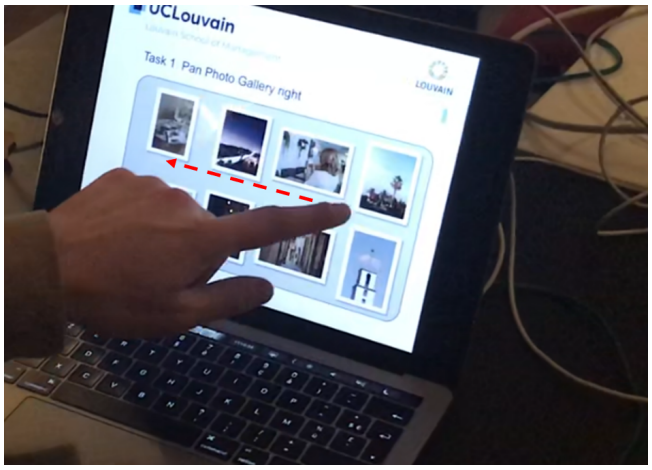


Figure B.1: Snapshot from the photo-browsing gesture elicitation study showing a participant performing a mid-air gesture.

B.1.3 Task

Participants were installed at a desk facing a screen, which presented them with *referents*, *i.e.,* common interactions with a photo or a video gallery, for which they proposed suitable gestures to execute those referents, *i.e.,* gestures that fit referents well while being easy to produce and remember. There were no constraints on the participants' choices of gestures, with the exception that a given gesture could not be assigned to more than one referent. The order of the referents was randomized per participant. Participants' fine motor skills and creativity levels were evaluated before the experiment, and a short questionnaire was administered after the experiment to give us an insight into the system user satisfaction.

B.1.4 Measures

We employed the same measures as Gheran *et al.* [42] to evaluate and understand users' preferences and cognitive and motor performance for gestures detected by a Kinect device:

1. We computed agreement rates (AR) for each Referent using the formula of Vatavu and Wobbrock [124] (p. 1327), as follows:

$$AR(r) = \frac{\sum_{i<j} \delta_{i,j}}{n \cdot (n-1)/2} \tag{B.1}$$

   where n is the number of participants from which gestures are elicited, and $\delta_{i,j}$ evaluates to 1 if the i-th and j-th participants are in agreement over referent r and to 0 otherwise.

2. Participants' creativity was evaluated before the elicitation procedure using a generic creativity test. The test is available online[1] and reports a score between 0 and 100 (higher scores denote more creativity) computed from answers to a set of 40 questions. We specifically chose this test due to its ease of application and large diversity of factors evaluated: *abstraction* (of concepts from ideas), *connection* (between things without an apparent link), *perspective* (shift in terms of space, time, and other people), *curiosity* (to change and improve things accepted as the norm), *boldness* (to push boundaries beyond accepted conventions), *paradox* (the ability to accept and work with contradictory concepts), *complexity* (the ability to operate with a large quantity of information), and *persistence* (to derive stronger solutions even when good ones have already been generated).

3. We measured participants' fine motor skills with a standard motor test of the NEPSY test batteries (a developmental NEuroPSYchological assessment) [57]. The test consists of touching each fingertip with the thumb of the same hand eight times in a row. Higher motor skills are reflected in less time to perform this task.

4. Thinking-Time measures the time, in seconds, needed by participants to propose a gesture for a given referent.

5. Goodness-of-Fit represents participants' subjective assessment, as a rating between 1 and 10, of their confidence about how well the proposed gestures fit the referents.

B.1.5 Taxonomy

To better understand the mid-air gesture proposals, we considered the following dimensions of analysis, strongly inspired by the taxonomy of Ring Gestures of Gheran *et al.* [42].

[1] http://www.testmycreativity.com/

1. *Scale* characterizes the amplitude of a gesture, with three categories: (a) large, (b) medium, and (c) small. These categories correspond respectively to arm, wrist, and fingers' range of movements.
2. *Laterality* characterizes how the two hands are employed to produce mid-air gesture, with four categories: (a) *dominant unimanual*, (b) *nondominant unimanual*, (c) *symmetric bimanual*, and (d) *asymmetric bimanual*. We adopted this dimension from Piumsomboon *et al.* [91], who used it to characterize users' mid-air gestures for AR applications.
3. *Nature* describes the meaning of a gesture, with four categories: (a) *symbolic*, (b) *metaphorical*, (c) *physical* and (d) *abstract*. Symbolic gestures depict popular symbols, such as the "thumbs up" gesture, that is often associated with liking something. Metaphorical gestures are gestures that mimic the use of a physical object, such as pushing a button on an imaginary remote control. Physical gestures manipulate the object or content directly, *e.g.,* rotating, or scaling it. Abstract gestures' mapping is arbitrary, there is no connection with symbols or physical objects. These categories are adopted from Ruiz *et al.* [99].
4. *Form* characterizes the relative importance of hand poses versus hand motion in the articulation of a mid-air gesture, with four categories: (a) *stroke*, (b) *static*, (c) *static with motion*, and (d) *dynamic*. The stroke category contains only gestures that consist of a single uninterrupted and fast movement. For static gestures, only the configuration of the hand (*i.e.,* the hand pose) is meaningful. Static with motion gestures consist of a succession of postures (*e.g.,* pinch two fingers to zoom out in a picture). Finally, in dynamic gestures, both the evolution of the hand configuration and the motion are important. This category was inspired by the taxonomy of Vatavu and Pentiuc [123].

B.2 Photo-Browsing

B.2.1 Experiment

This section will add more details about the experiment that were omitted at the beginning of this chapter.

Participants

Thirty (30) participants (16 female), aged between 18 and 77 years old ($M = 34.5$, $SD = 17.5$ years), volunteered for the study. Participant occupations included teachers, students, office clerks, computer scientists, retirees. 93% (28/30) of the participants were right-handed. All but one of the participants regularly used a smartphone or a tablet device and, thus, were accustomed to touch and gesture input.

Design

The study was within-subjects with one independent variable: the Referent. It is a nominal variable with 31 conditions, representing common tasks to execute in a collection-of-pictures type of environment, such as a photo gallery, with a Kinect device: (1) pan the photo gallery right, (2) left, (3) up and (4) down, (5) go to the next and (6) previous page of the photo gallery, (7) zoom in and (8) out in a photo, (9) maximize a photo from normal view to full screen and (10) unmaximize it, (11) open and (12) close photo information, (13) set the layout to carousel, (14) start and (15) stop a slideshow, (16) take a selfie, (17) insert a new photo, (18) duplicate

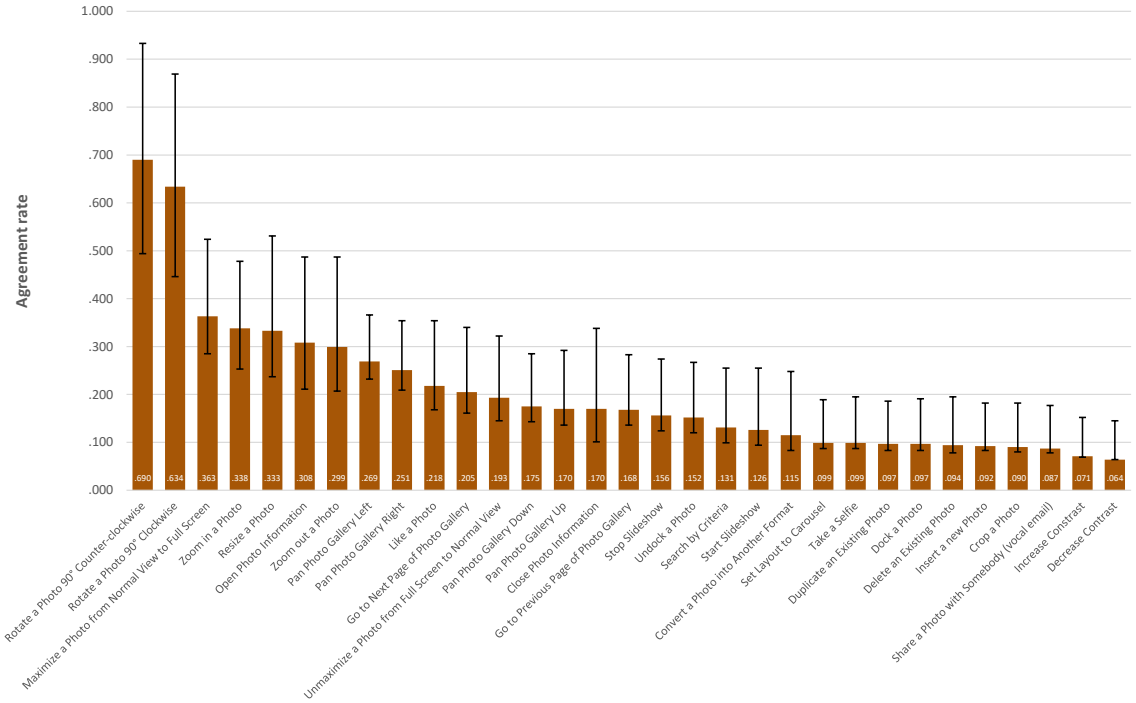


Figure B.2: Agreement rates for the gesture proposals (photo-browsing). Notes: referents are ordered on the horizontal axis in descending order of their agreement rates; error bars show 95% CIs computed with the AGATe tool [124].

and (19) delete an existing photo, (20) crop and (21) resize a photo, (22) rotate a photo 90° clockwise and (23) 90° counter-clockwise, (24) increase and (25) decrease the contrast of a photo, (26) dock and (27) undock a photo, (28) like a photo, (29) search photos within the gallery by criteria, (30) share a photo with somebody, (31) convert a photo into another format.

The choice of referents represents a collection of basic (e.g., pan the photo gallery right/left/up/down) and more complex tasks (e.g., search photos within the gallery by criteria) likely to be useful for interactions with a gallery of pictures.

B.2.2 Results

We collected a total number of 930 gesture proposals from 30 (participants) × 31 (referents) conditions, which we clustered into groups of similar gesture types according to the following criteria: We did not distinguish gestures according to their scale (*i.e.,* whether the amplitude of the gesture is large or small), direction, or handedness (*i.e.,* whether the gesture was drawn with the dominant or non-dominant hand). However, we considered gestures executed with different hand poses as different, meaning, *e.g.,* a "swipe" with the index finger pointed is considered different from the same "swipe" with the hand open.

We identified 21 distinct gestures. Next, we analyze the consensus between participants' gesture proposals and assign gesture types to meaningful categories.

Consensus Between Proposed Gestures

Figure B.2 shows the agreement rates obtained for each Referent. Overall, agreement rates are average, between .064 and .690 (M=.205, SD=.147). These results fit within the reported agreement rates in the literature of gesture elicitation; see Vatavu and Wobbrock [124] (p. 1332) that summarize agreement rates of 18 studies. According to the recommendations of Vatavu

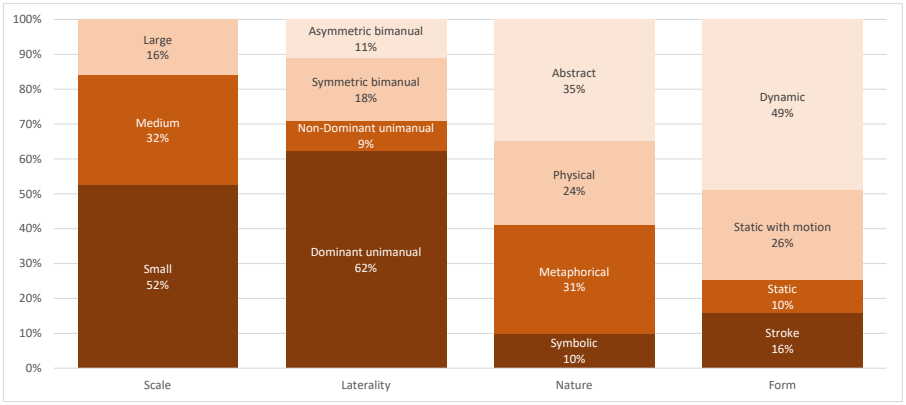


Figure B.3: Observed percentages of gestures for each category of our taxonomy (photo-browsing).

and Wobbrock [124] to interpret the magnitudes of agreement rates, most of our results fall inside the medium consensus (.100 − .300) category. These results were confirmed by Kendall's coefficient of concordance:[2] $W = .150$ ($\chi^2(30) = 134.576$, $p < .0001$). As Kendall's coefficient is related to the average of Spearman correlation coefficients between pairs of rankings [51] (p. 276), we can interpret the magnitude of its effect as small (*i.e.,* close to .100) according to Cohen's suggested limits for interpreting effect sizes.

Taxonomy of Mid-Air Gestures

Figure B.3 illustrates the observed percentages of gestures falling in each category according to the taxonomy presented at the beginning of this chapter.

Gestures' Goodness of Fit

Participants rated their gesture proposals with numbers from 1 (poor fit) to 10 (excellent fit) to denote their confidence in the goodness of fit of their proposals. Goodness-of-Fit correlated significantly with the Agreement-Rate (Pearson's $r_{(N=31)}$=.530, R^2=.281, p=.002): referents that reached high agreement rates were assigned gestures that were rated a good fit. Figure B.4 shows this relationship between the goodness of fit and agreement rates.

Thinking Time

The average thinking time was quite high ($M = 8.58\,\text{s}$, $SD = 10.63\,\text{s}$). Figure B.5 shows the relationship between thinking times and agreement rates per Referent. Thinking-Time correlated significantly with Agreement-Rate (Pearson's $r_{(N=31)} = -.409$, $R^2 = .167$, $p = .022$): the agreement rate decreased for longer thinking times. Thinking time seemed to be longer for more complex tasks (*e.g.,* search by criteria or convert a photo into another format), and shorter for simpler tasks (*e.g.,* pan the photo gallery left/right or rotate a photo 90° clockwise). We noted that a majority of the simpler task already had a well-known gesture assigned to them for touchscreen devices. As most of our participants actively used a smartphone or a tablet, this may explain the lower thinking time and higher agreement rate.

[2]Kendall's W is a normalization of the Friedman statistic used to assess the agreement between multiple raters with a number ranging between 0 (no agreement at all) and 1 (perfect agreement).

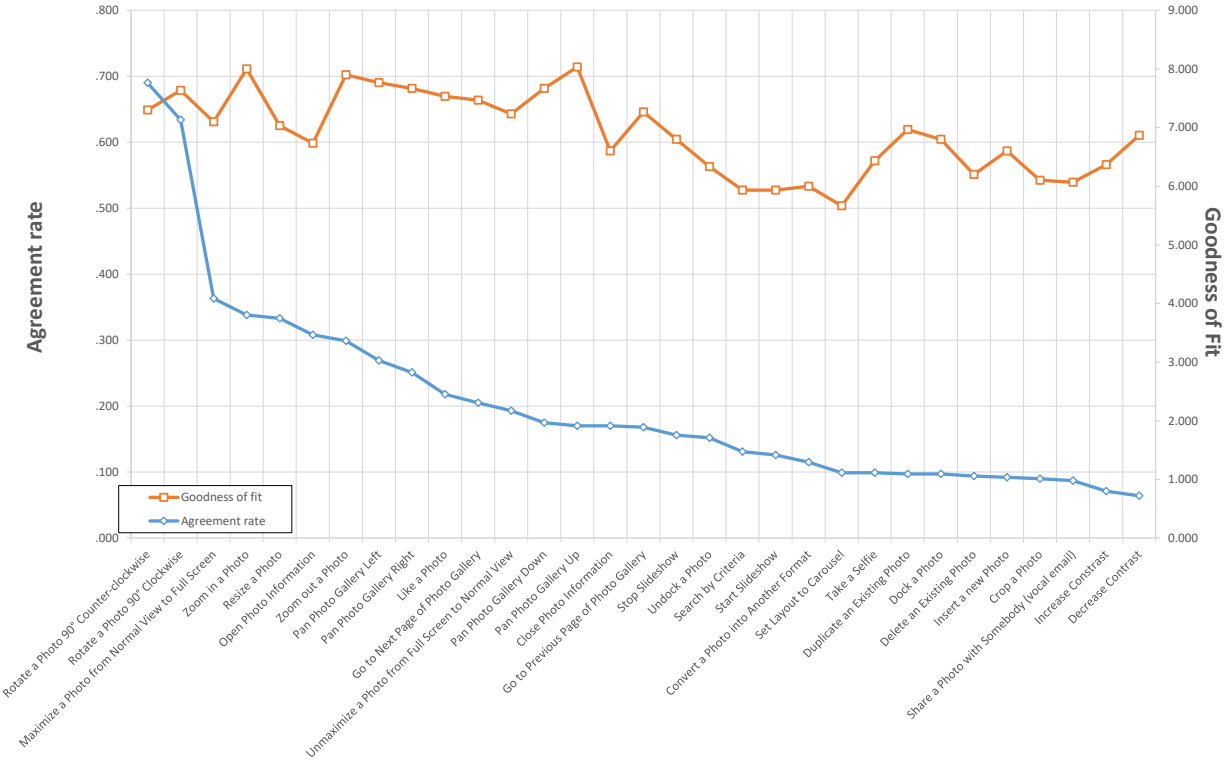


Figure B.4: Relationship between Agreement-Rate and Goodness-of-Fit for each referent (photo-browsing).

B.3 Video-Browsing

B.3.1 Experiment

This section will add more details about the experiment that were omitted at the beginning of this chapter.

Participants

Twenty-two (22) participants (8 female), aged between 15 and 54 years old ($M = 28.9$, $SD = 12.6$ years), volunteered for the study. Participant occupations included students, economists, teachers, engineers, and midwives. All the participants regularly used computers, and 90.9% of the participants (20/22) regularly used either a smartphone or a tablet device.

Design

The study was within-subjects with one independent variable: the Referent. It is a nominal variable with 27 conditions, representing common tasks to execute in a collection-of-videos type of environment, such as a video gallery, with a Kinect device: (1) pan the video gallery right, (2) left, (3) up and (4) down, (5) go to the next and (6) previous page of the video gallery, (7) zoom in and (8) out in a video, (9) maximize a video from normal view to full screen and (10) unmaximize it, (11) start and (12) stop playing a video, (13) go at the beginning and (14) the end of a video, (15) open and (16) close video information, (17) set the layout to carousel, (18)

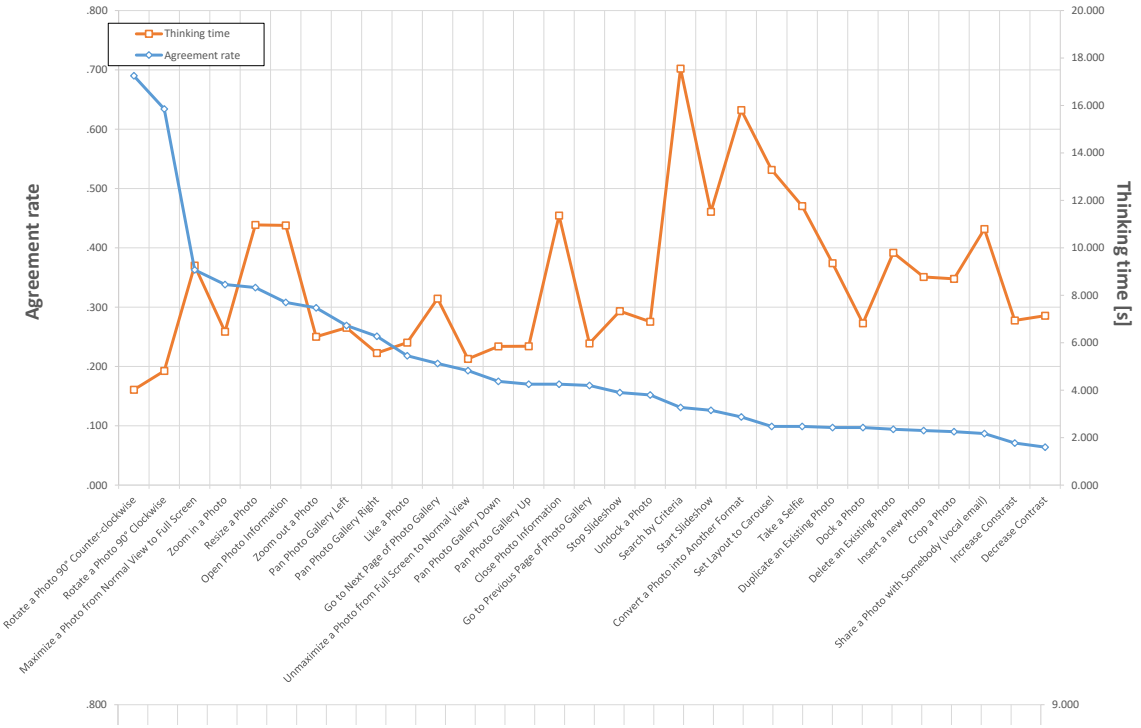


Figure B.5: Relationship between AGREEMENT-RATE and THINKING-TIME (photo-browsing).

insert a new video, (19) duplicate and (20) delete an existing video, (21) rotate a video 90° clockwise and (22) 90° counter-clockwise, (23) dock and (24) undock a video, (25) like a video, (26) search videos within the gallery by criteria, (27) share a video with somebody.

The choice of referents represents a collection of basic (*e.g.,* zoom in/out in a video) and more complex tasks (*e.g.,* share a video with someone) likely to be useful for interactions with a collection of videos.

B.3.2 Results

We collected a total number of 594 gesture proposals from 22 (participants) × 27 (referents) conditions, which we clustered into groups of similar gesture types according to the following criteria: We distinguished gestures according to their hand pose and the axis in which they were executed (*e.g.,* parallel to the body, or parallel to the normal vector to the body), but we did not distinguish them according to their scale, direction, or handedness. We identified 23 distinct gestures. Next, we analyze the consensus between participants' gesture proposals and assign gesture types to meaningful categories.

Consensus Between Proposed Gestures

Figure B.6 shows the agreement rates obtained for each REFERENT. Overall, agreement rates are average, between .052 and .593 (M=.229, SD=.162). These results fit within the reported agreement rates in the literature of gesture elicitation; see Vatavu and Wobbrock [124] (p. 1332) that summarize agreement rates of 18 studies. According to the recommendations of Vatavu and Wobbrock [124] to interpret the magnitudes of agreement rates, most of our results fall inside the medium consensus (.100 − .300) category.

These results were confirmed by Kendall's coefficient of concordance:[3] $W = .251$ ($\chi^2(26) =$

[3]Kendall's W is a normalization of the Friedman statistic used to assess the agreement between

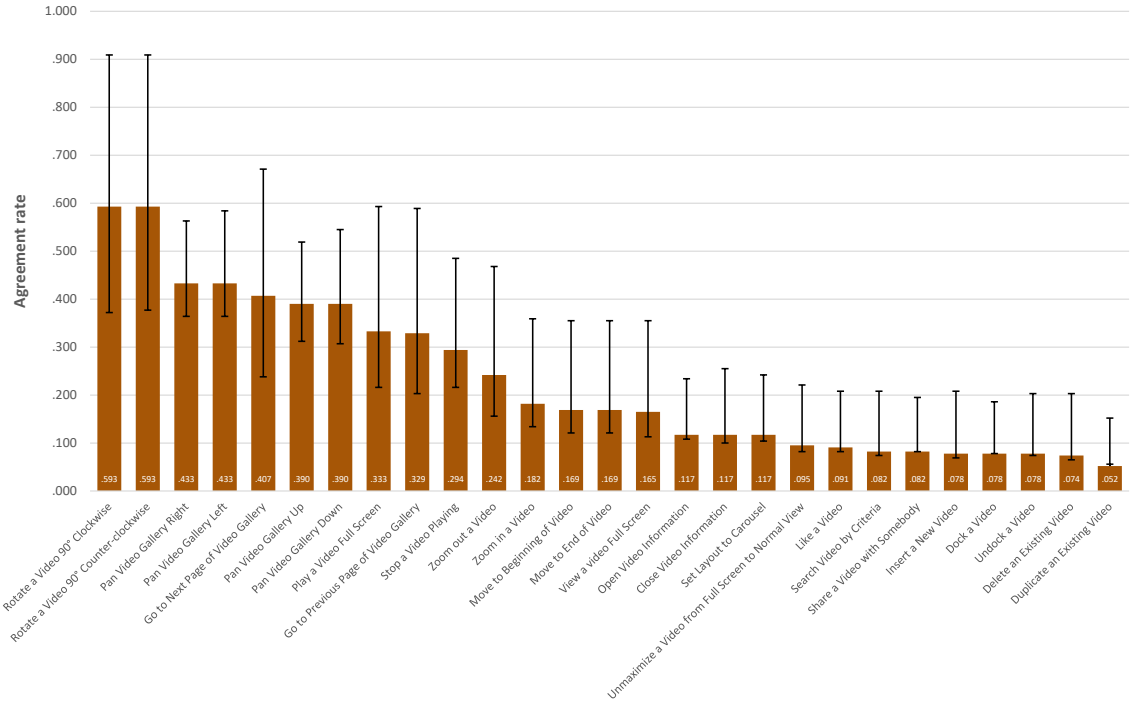


Figure B.6: Agreement rates for the gesture proposals (video-browsing). Notes: referents are ordered on the horizontal axis in descending order of their agreement rates; error bars show 95% CIs computed with the AGATe tool [124].

143.446, $p < .0001$). As Kendall's coefficient is related to the average of Spearman correlation coefficients between pairs of rankings [51] (p. 276), we can interpret the magnitude of its effect as medium (*i.e.*, close to .300) according to Cohen's suggested limits for interpreting effect sizes.

Taxonomy of Mid-Air Gestures

Figure B.7 illustrates the observed percentages of gestures falling in each category according to the taxonomy presented at the beginning of this chapter.

Gestures' Goodness of Fit

Participants rated their gesture proposals with numbers from 1 (poor fit) to 10 (excellent fit) to denote their confidence in the goodness of fit of their proposals. Goodness-of-Fit correlated significantly with the Agreement-Rate (Pearson's $r_{(N=27)}$=.723, R^2=.522, p<.0001): referents that reached high agreement rates were assigned gestures that were rated a good fit. Figure B.8 shows this relationship between the goodness of fit and agreement rates.

Thinking Time

The average thinking time was quite high (M = 9.85 s, SD = 26.90 s). Figure B.9 shows the relationship between thinking times and agreement rates per Referent. Thinking-Time correlated significantly with Agreement-Rate (Pearson's $r_{(N=27)} = -.545$, $R^2 = .297$, $p = .0033$): the agreement rate decreased for longer thinking times. Usually, simpler tasks (*e.g.*, rotate a video, or zoom in/out) were associated with shorter thinking time. Most of these simple tasks were already associated with well-known gestures (mostly from touchscreen devices).

multiple raters with a number ranging between 0 (no agreement at all) and 1 (perfect agreement).

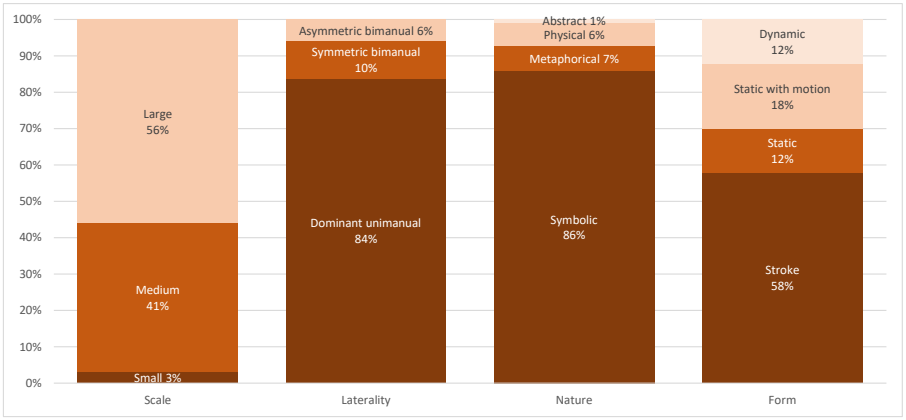


Figure B.7: Observed percentages of gestures for each category of our taxonomy (video-browsing).

B.4 What Can We Conclude From Both Experiments?

In this section, we analyze the (dis)similarities between the results of the photo-browsing and the video-browsing elicitation studies. We try to draw conclusions for future elicitation studies.

B.4.1 Agreement Rate, Thinking Time and Goodness of Fit

Both experiments showed that thinking time correlated significantly with the agreement rate: a higher thinking time seemed linked to a lower agreement rate. The same can be said for the goodness of fit: the agreement rate increased with the goodness of fit in both elicitation studies.

Influence of the Gesture Pairing Criteria

Both experiments showed quite high agreement rates. Gheran *et al.* [42] have shown that the gesture pairing criteria may influence the magnitude of the agreement rate. In both experiments, the criteria used to categorize gestures were quite relaxed (e.g., gestures executed in different directions were considered equivalent). This can explain their slightly higher agreement rates compared to other similar experiments [124].

Influence of the Methodology and Experiment Design

The visual indication given to the users for each referent may have impacted their choice of gesture. Especially when those visual indications were not extremely clear. Furthermore, having to choose a unique gesture for each referent may have negatively impacted the agreement rate, as participants sometimes had to propose a different gesture from the one that they would have preferred. The setup for the experiment may also have impacted the choice of gestures of the user. Indeed, in the photo-browsing experiment, participants were situated very close to the screen showing them the visual indications, to the point where some of them physically touched the screen while performing their gestures. This may have encouraged them to choose more "touch-oriented" gestures. It could be interesting to repeat the experiment in a more "open" setup, where participants are installed further away from the screen.

B.4.2 Taxonomy of Gestures

Despite the referents of the two experiments being very similar, we observed quite varying percentages of gestures falling into the categories of our taxonomy (see figures B.3 and B.7):

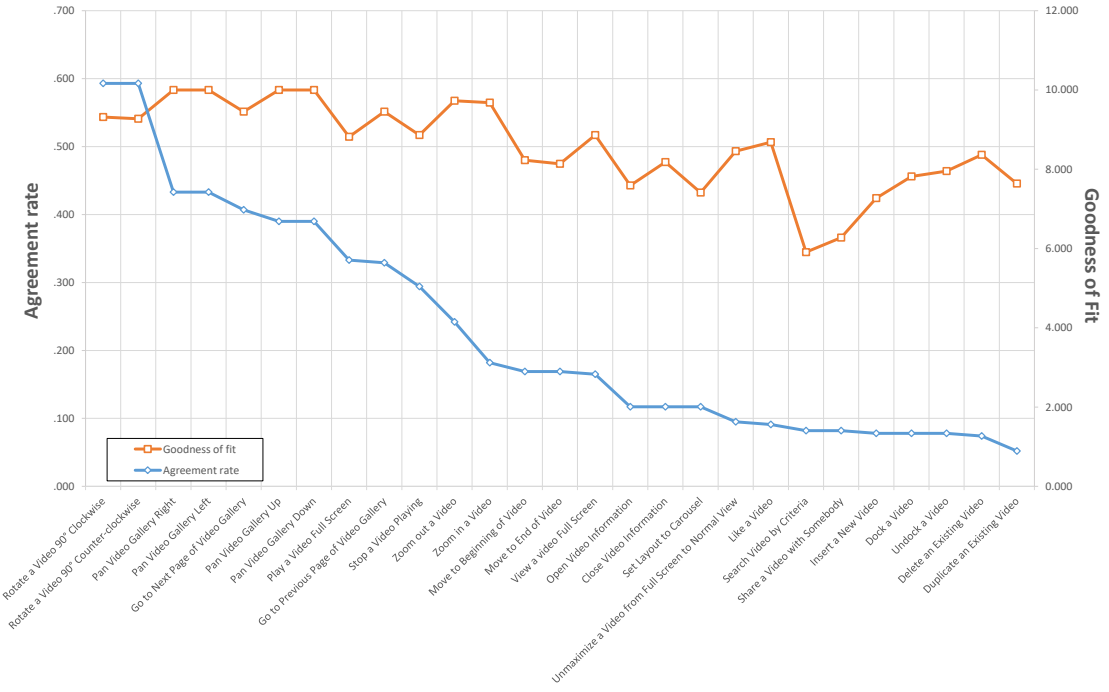


Figure B.8: Relationship between Agreement-Rate and Goodness-of-Fit for each referent (video-browsing).

- *Scale*: in the photo-browsing experiment, a majority of gestures fall into the *small* (52%) and *medium* (32%) categories, while respectively 56% and 41% of the gestures elicited in the video-browsing experiment fall in the *large* and *medium* categories. These conflicting results may be due to differences in the setup of the experiments. Participants of the photo-browsing experiment were situated close enough to the screen that they could touch it. This may have encouraged them to use more touchscreen-inspired gestures and to limit the amplitude of their movements.

- *Laterality*: the results of both elicitation studies seem quite similar, with a strong preference for *dominant unimanual* gestures, although the photo-browsing experiment showed a higher percentage of gestures falling into the other categories.

- *Nature*: the distribution of gestures into the various categories is vastly different between both experiments. In the video-browsing elicitation study, most gestures are *symbolic* (86%), while they are more evenly distributed into the 4 categories in the photo-browsing experiment, with a majority of them being either *abstract* (35%) or *metaphorical* (31%).

- *Form*: both elicitation studies show a similar number of gestures falling into the *static* and *static with motion* categories but they disagree on the *Dynamic* (12% vs 49% for resp. the video-browsing and the photo-browsing experiment) and *Stroke* categories (58% vs 16% for resp. the video-browsing and the photo-browsing experiment).

B.4.3 Guidelines for Future Elicitation Studies

In this section, we discuss the results of our experiment, we suggest elements to investigate in future elicitation studies and we derive some recommendations for mid-air gestures.

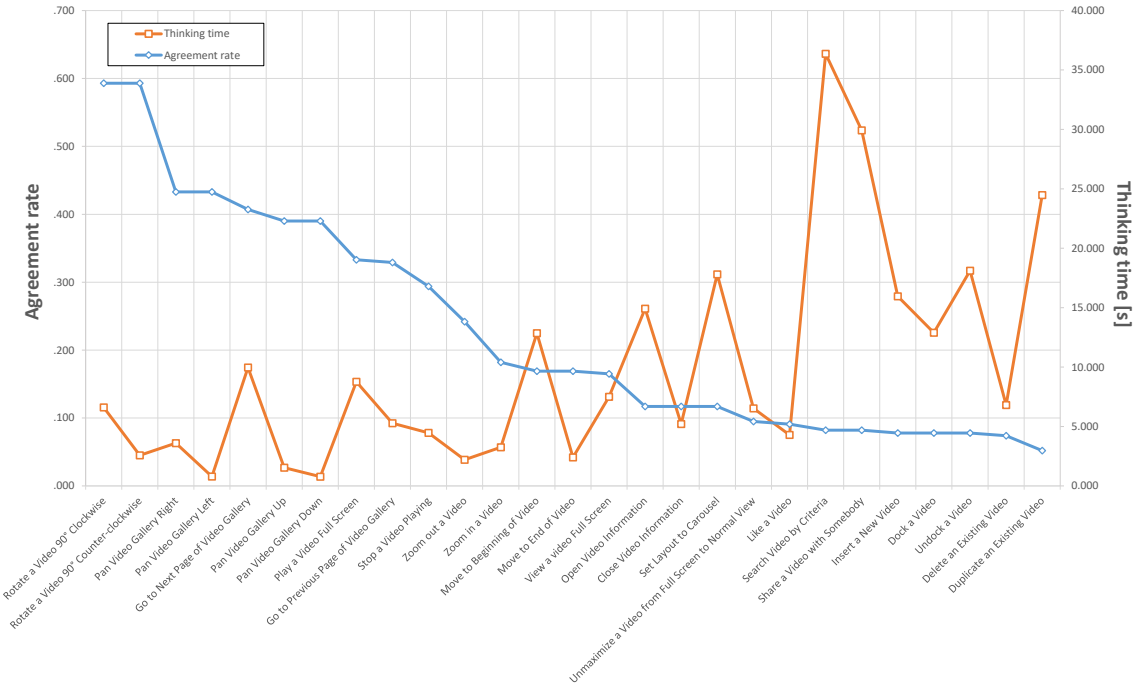


Figure B.9: Relationship between Agreement-Rate and Thinking-Time (video-browsing).

- *Motion seems to be preferred over static gestures.*
  Only about a tenth of the proposed gestures were static, with the rest of the gestures being either dynamic, static with motion, or stroke. Participants especially avoided static gestures for referents representing "navigation" actions (*e.g.,* panning the photo gallery left or right). However, they were more frequently used for actions such as like a photo/video, set the layout to carousel, or take a selfie.

- *Users prefer gestures of the dominant hand.*
  As discussed before, most of the proposed gestures involved only the dominant hand, with a minority of gestures involving two hands. However, there was no real consensus between both experiments on the size of those gestures.

- *Many proposed gestures are inherited from touchscreen devices.*
  Participants mostly limited themselves to using "touch-inspired" gestures, such as swiping with a hand/finger or tapping with one finger. As opposed to the findings of Gheran *et al.* [42] when studying gestures for smart rings, very few of the proposed gestures involved more complex gestures, such as using a hand as an invisible "tv-remote", that are not executable on touchscreen devices. Maybe wearing or physically holding an input device (e.g., a smart ring) encourages participants to find more original gestures, by giving them a hint about the kind of gestures they could execute.

- *User interface plays an important role while defining intuitive gestures.*
  The agreement rate for some referents (e.g., search by criteria, share a video with someone) was very low. These referents mainly represented actions that are quite complex to execute on devices with touch or mouse-input. For instance, sharing a video with a contact usually implies selecting a video, the sharing medium, and finally the intended recipient. Without a proper hint of the layout of the interface, devising a gesture that could replace all these actions at once is difficult as it mostly depends on how the interface is setup. Future

experiments should concentrate more on designing an interface that would make such actions easy and natural to perform through mid-air gestures.

- *Consider context-sensitive gestures.*
  The constraint that a unique gesture should be assigned to each referent was quite frustrating according to some participants. To reduce this frustration, we could assign a gesture to multiple referents if the context allows the system to correctly recognize the action prompted by the gesture (*e.g.,* a flick to the left in the photo gallery could have a different effect than the same gesture when viewing a photo in full screen). This would limit the number of gestures to be remembered, and thus make it easier for new users to learn how to navigate through the interface.

Referent	AR	Most frequent	Second most frequent
1. *Pan Photo Gallery Right*	.251	1 hand flick	Multiple fingers swipe
2. *Pan Photo Gallery Left*	.269	Multiple fingers swipe	1 finger swipe
3. *Pan Photo Gallery Up*	.170	1 finger swipe	Hand rotation
4. *Pan Photo Gallery Down*	.175	1 hand flick	1 finger swipe
5. *Go to Next Page of Photo Gallery*	.205	Multiple fingers swipe	1 hand flick
6. *Go to Previous Page of Photo Gallery*	.168	Multiple fingers swipe	1 hand flick
7. *Zoom in a Photo*	.338	Splay the hand	Bring hands further apart
8. *Zoom out a Photo*	.299	Clench the hand	Bring hands further apart
9. *Maximize a Photo from Normal View to Full Screen*	.363	Bring hands further apart	Splay the hand
10. *Unmaximize a Photo from Full Screen to Normal View*	.193	Clench the hand	Bring hands closer
11. *Open Photo Information*	.308	1 finger tap	Splay the hand
12. *Close Photo Information*	.170	1 finger tap	1 finger swipe
13. *Set Layout to Carousel*	.099	Hand rotation	1 finger tap
14. *Start Slideshow*	.126	1 finger tap	Hand pose
15. *Stop Slideshow*	.156	1 finger tap	Move flat hand forward/backward
16. *Take a Selfie*	.099	Hand pose	1 finger tap
17. *Insert a new Photo*	.092	Hand pose	1 finger tap
18. *Duplicate an Existing Photo*	.097	1 finger swipe	1 finger tap
19. *Delete an Existing Photo*	.094	1 finger swipe	Hand pose
20. *Crop a Photo*	.090	Draw	Bring hands further apart
21. *Resize a Photo*	.333	Bring hands further apart	Splay the hand
22. *Rotate a Photo 90° Clockwise*	.634	Hand rotation	N/A
23. *Rotate a Photo 90° Counterclockwise*	.690	Hand rotation	N/A
24. *Increase Constrast*	.071	1 finger swipe	Open arms (with flat hands)
25. *Decrease Contrast*	.064	1 finger swipe	Multiple fingers swipe
26. *Dock a Photo*	.097	1 finger swipe	1 hand flick
27. *Undock a Photo*	.152	1 finger swipe	Take and throw
28. *Like a Photo*	.218	1 finger tap	Hand pose
29. *Search by Criteria*	.131	Hand pose	Draw
30. *Share a Photo with Somebody (vocal email)*	.087	Hand pose	1 finger swipe
31. *Convert a Photo into Another Format*	.115	Bring hands further apart	1 finger tap

References with agreement rates above average are highlighted in blue.

Table B.1: First and second most frequent gesture proposals for each referent (photo-browsing).

Referent	AR	Most frequent	Second most frequent
1. *Pan Video Gallery Right*	.433	1 hand drag	1 finger drag
2. *Pan Video Gallery Left*	.433	1 hand drag	1 finger drag
3. *Pan Video Gallery Up*	.390	1 hand drag	1 finger drag
4. *Pan Video Gallery Down*	.390	1 hand drag	1 finger drag
5. *Go to Next Page of Video Gallery*	.407	1 hand drag	1 finger drag
6. *Go to Previous Page of Video Gallery*	.329	1 hand drag	1 finger drag
7. *Zoom in a Video*	.182	1 hand drag	Close one hand
8. *Zoom out a Video*	.242	1 hand drag	Close one hand
9. *Set a Video in Full Screen*	.165	1 finger tap	Drag 1 finger from each hand
10. *Unmaximize a Video from Full Screen to Normal View*	.095	Drag 1 finger from each hand	2 hands drag
11. *Play a Video*	.333	1 finger tap	1 finger drag
12. *Stop Playing a Video*	.294	Push/pull 1 hand	1 finger tap
13. *Move to the Beginning of Video*	.169	1 hand drag	1 finger tap
14. *Move to the End of a Video*	.169	1 hand drag	1 finger tap
15. *Open Video Information*	.117	1 finger tap	Push/pull 1 hand
16. *Close Video Information*	.117	1 hand drag	1 finger drag
17. *Set Layout to Carousel*	.117	1 hand drag	Tap and drag 1 finger
18. *Insert a New Video*	.078	Push/pull and drag 1 hand	Push/pull 1 hand
19. *Duplicate a Video*	.052	2 hands drag	Tap and drag 1 finger from each hand
20. *Delete a Video*	.074	1 hand drag	1 finger drag
21. *Rotate a Video 90° Clockwise*	.593	1 hand drag	Push/pull 1 hand
22. *Rotate a Video 90° Counter-clockwise*	.593	1 hand drag	Push/pull 1 hand
23. *Dock a Video*	.078	Push/pull and drag 1 hand	Tap and drag 1 finger
24. *Undock a Video*	.078	Push/pull and drag 1 hand	Close, push/pull and drag 1 hand
25. *Like a Video*	.091	1 finger drag	Push/pull 1 hand
26. *Search for a Video by Criteria*	.082	1 finger drag	1 hand drag
27. *Share a Video with Somebody*	.082	1 hand drag	Close, push/pull and drag 1 hand

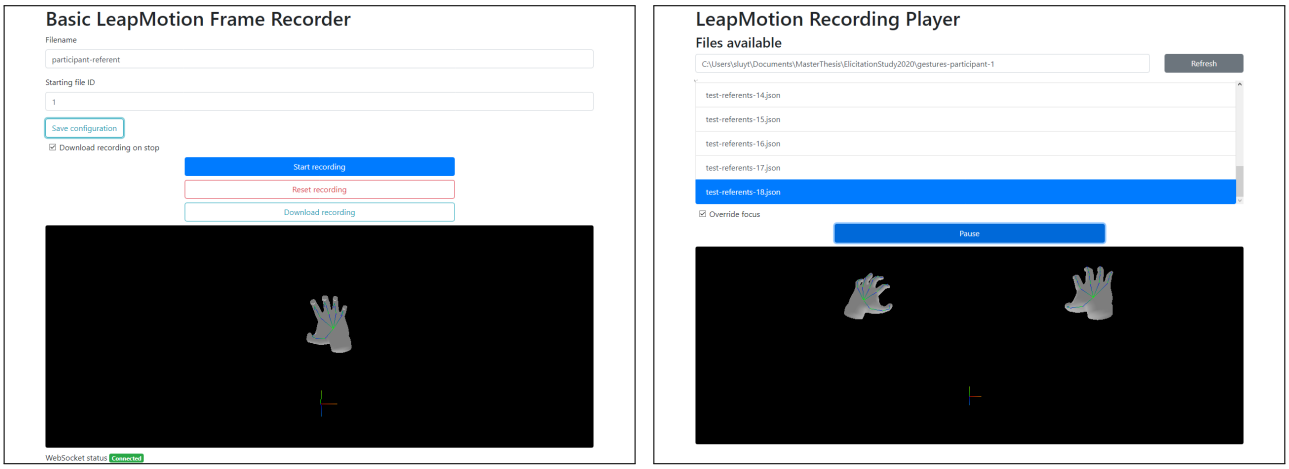
Referents with agreement rates above average are highlighted in blue.

Table B.2: First and second most frequent gesture proposals for each referent (video-browsing).

Appendix C

Recording and Playback Tool

For the purposes of our new elicitation study, it was important to have a simple and reliable tool to both record and playback the mid-air gestures proposed by each participant. As no software with the required set of features existed, a new one was designed and is now openly available on GitHub[1].



(a) Recorder (b) Player

Figure C.1: Screenshots of the player and recorder tools.

The recording tool (Figure C.1a) retrieves tracking data by connecting to the LMC's WebSocket Server[2]. It offers the following functionalities:

- Select a base name that stays the same for all the recordings and a starting index, which is incremented for each new recording.
- Start/stop/reset/download recording.
- Automatically download the recording after stopping it. This functionality proved very useful to streamline the process of recording gestures during the elicitation study.
- Visualize the data, *e.g.*, to check whether the LMC correctly picked up participants' gestures.

The playback tool (Figure C.1b) emulates an LMC. It consists of a WebSocket server that

[1] https://github.com/sluyters/LeapGesturePlayback

[2] https://developer-archive.leapmotion.com/documentation/javascript/supplements/Leap_JSON.html

keeps track of the state of clients and sends them recorded frames. It offers the following functionalities:

- Select a recording from anywhere on the computer.
- Play/pause recording.
- Focus override: send the frames to each known client, regardless of whether they are focused or not. If this option is not enabled, frames are only sent to focused clients (*i.e.,* those that the user is actively using).
- Visualize the recording, *e.g.,* to analyze participants' gesture propositions after the completion of the elicitation study.

Appendix D

New Elicitation Study

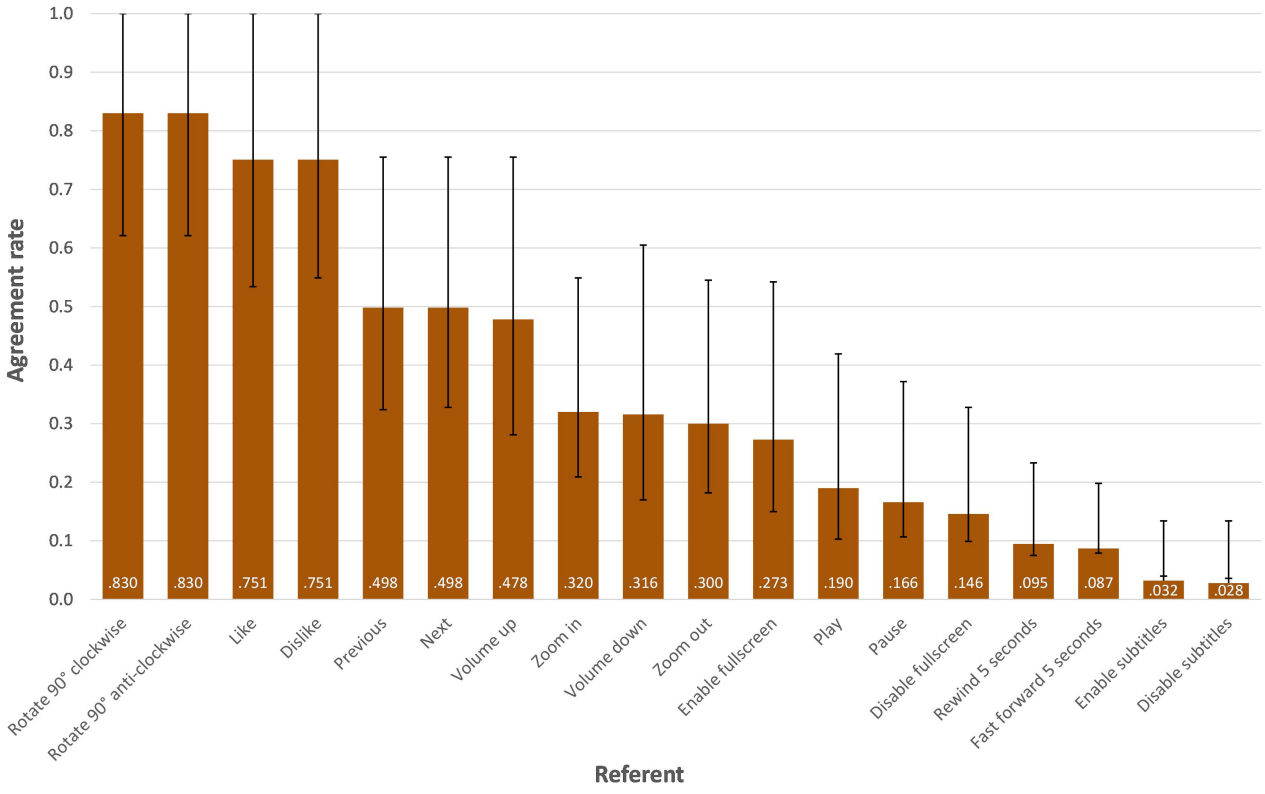


Figure D.1: Agreement rates for the gesture proposals. Notes: referents are ordered on the horizontal axis in descending order of their agreement rates; error bars show 95% CIs computed with the AGATe tool [124].

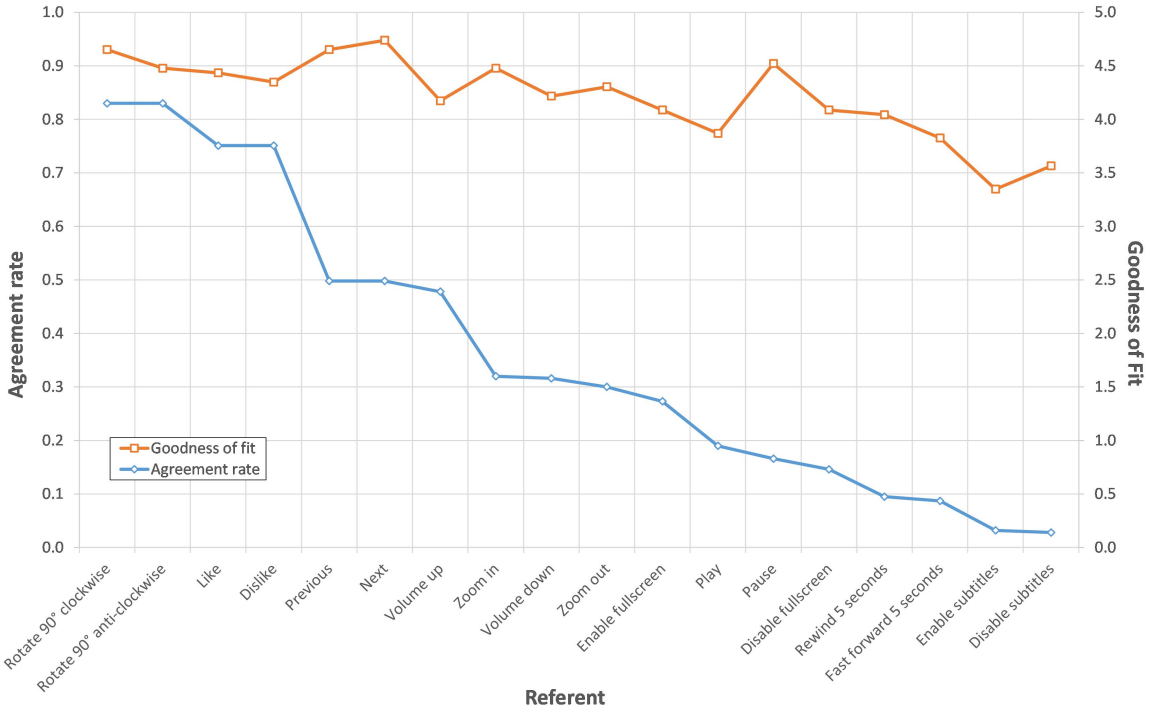


Figure D.2: Relationship between Agreement-Rate and Goodness-of-Fit.

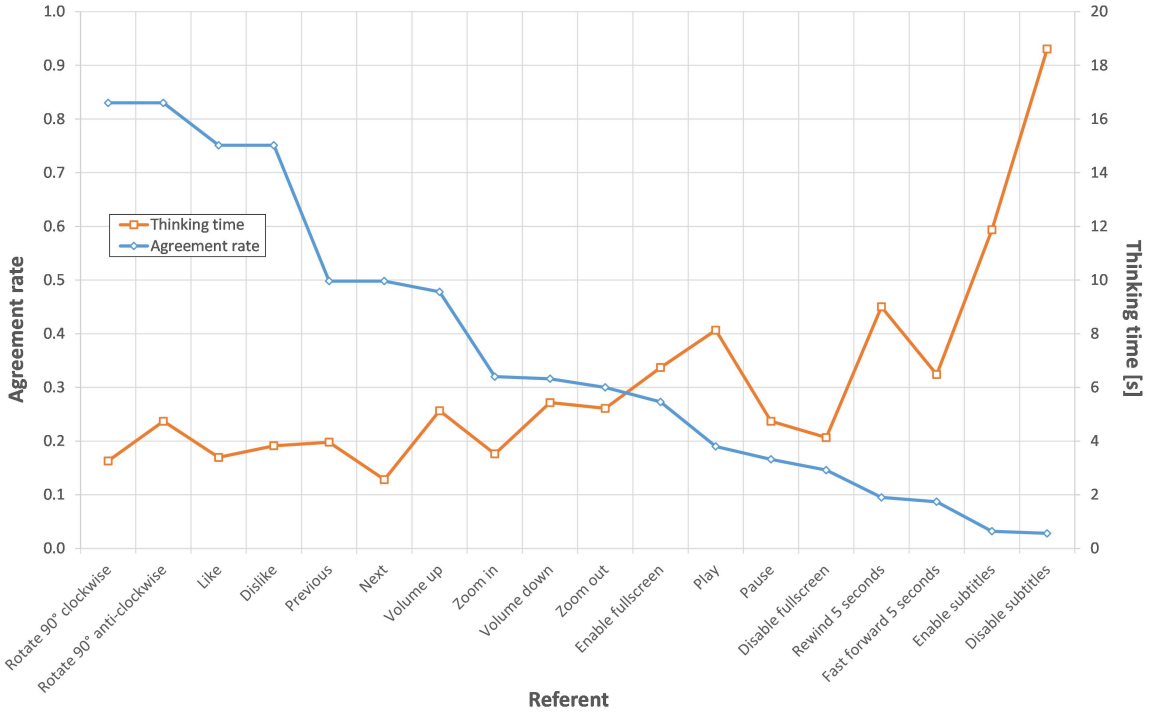


Figure D.3: Relationship between Agreement-Rate and Thinking-Time.

Appendix E

LUI Gesture Set

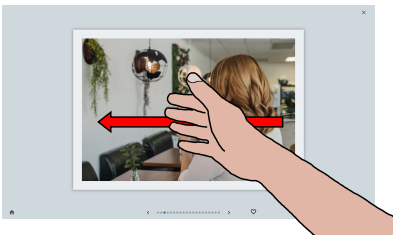
Action	Gesture	Justifications and Notes
Previous	Flick right	We kept the most popular gestures, but changed the *swipe* motion into the similar but faster *flick* motion.
Next	Flick left	
Enable fullscreen	Tap	We defined a new gesture, because in the LUI project, enabling fullscreen requires users to first select the item that they want to magnify. However, neither of the two most popular elicited gestures (*bring hands further apart* and *pinch out*) were appropriate for this task.
Disable fullscreen	Flick up	As neither of the two most popular elicited gestures were ideal for our context of use, we selected a different one. Indeed, the most popular gesture involved two hands, which may have proven harder to recognize and would have required users to have two free hands. The second most popular gesture, *pinch in*, would have clashed with the *zoom in* action.
Zoom in	Pinch out	We kept the most popular gestures from the elicitation study.
Zoom out	Pinch in	
Volume up	Swipe up*	
Volume down	Swipe down*	
Rotate clockwise	Rotate a knob clockwise	
Rotate anti-clockwise	Rotate a knob anti-clockwise	
Play	Tap	
Pause	Tap	
Like	Thumbs up	
Dislike	Thumbs down	
Fast-forward	Swipe left*	We selected the second most popular elicited gesture in order to stay coherent with the *rewind* action. Furthermore, the most popular gesture (*flick right twice*) would have been problematic to recognize, as it can clash with both *flick left* and *flick right*.
Rewind	Swipe right*	We kept the most popular elicited gesture.

* The *swipe up*, *swipe down*, *swipe left* and *swipe right* gestures must be performed while pointing the index finger in order to differentiate more easily between them and the *flick up*, *flick left* and *flick right* gestures (see Figure E.1 for graphical representations of these gestures).

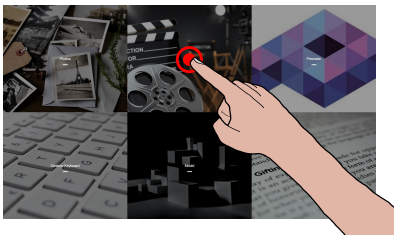
Table E.1: Assigned gestures for each referent.



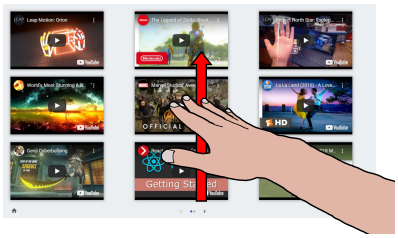
(a) Previous



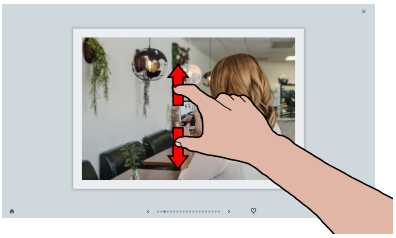
(b) Next



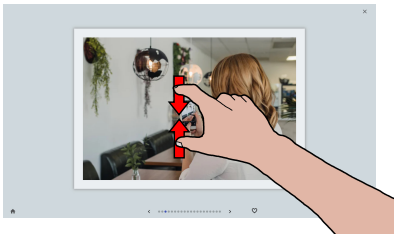
(c) Enable fullscreen



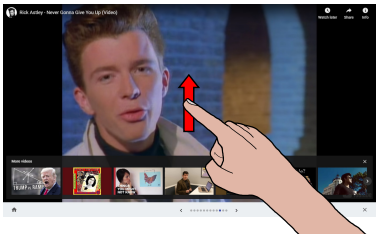
(d) Disable fullscreen



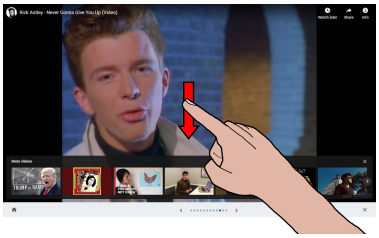
(e) Zoom in



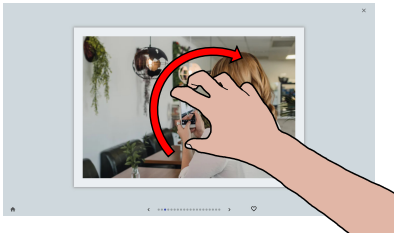
(f) Zoom out



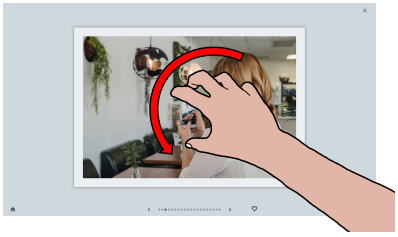
(g) Volume up



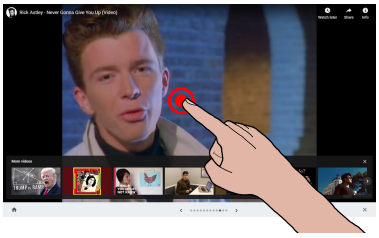
(h) Volume down



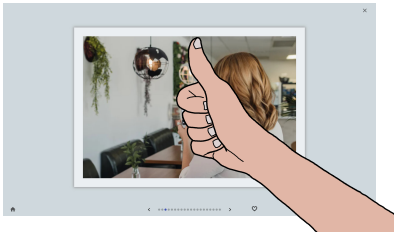
(i) Rotate clockwise



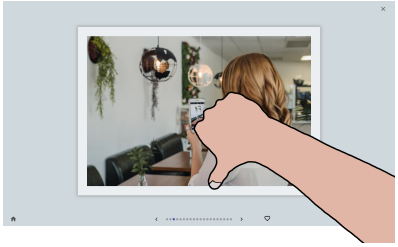
(j) Rotate anti-clockwise



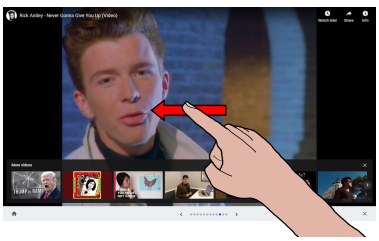
(k) Play/pause



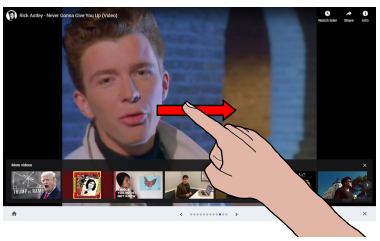
(l) Like



(m) Dislike



(n) Fast-forward



(o) Rewind

Figure E.1: Illustrations of each action and its assigned gesture.

Appendix F

Code of the Software

This appendix contains most of the code of the framework and the LUI application.

F.1 Framework

F.1.1 Frozen Spots

WebSocket Server

index.js

```
const WebSocket = require('ws');
const http = require('http');
const config = require('./config');
const FrameProcessor = require('./framework/frame-processor').FrameProcessor;

// Load the modules
const Sensor = config.sensor.module;

// Main function
function run() {
    // Initialize the sensor interface, dataset, recognizer and segmenter
    var sensor = new Sensor(config.sensor.options);
    var frameProcessor = new FrameProcessor(config);
    // Start the websocket server
    var wsServer = getWebSocketServer(config.server.ip, config.server.port);
    wsServer.on('connection', async function connection(ws) {
        console.log('Connected!')
        // Handle messages from the client
        ws.on('message', function(message) {
            if (config.general.debug) {
                console.log(message)
            }
            var msg = JSON.parse(message);
            if (msg.type === 'operation') {
                for (const operation of msg.data) {
                    if (operation.type === 'addPose') {
                        frameProcessor.enablePose(operation.name);
```

```
                    } else if (operation.type === 'addGesture') {
                        frameProcessor.enableGesture(operation.name);
                    } else if (operation.type === 'removePose') {
                        frameProcessor.disablePose(operation.name);
                    } else if (operation.type === 'removeGesture') {
                        frameProcessor.disableGesture(operation.name);
                    }
                }
            }
        });
        // Process sensor frames
        sensor.loop((frame, appData) => {
            let message = getMessage('data');
            if (appData && config.general.sendContinuousData) {
                // If there is continuous data to send to the application
                message.data.push({
                    'type': 'frame',
                    'data': appData
                })
            }
            // Gesture recognition
            var ret = frameProcessor.processFrame(frame);
            if (ret) {
                // If there is gesture data to send to the application
                message.data.push(ret);
                if (config.general.debug) {
                    console.log(JSON.stringify(message));
                }
            }
            if (message.data.length > 0) {
                ws.send(JSON.stringify(message));
            }
        });
        // Stop processing frames after disconnection
        ws.on('close', function() {
            console.log("Disconnected!")
            sensor.stop();
            frameProcessor.resetContext();
        });
        // Connection error
        ws.on('error', function(error) {
            console.log(JSON.stringify(error));
        });
    });
}

function getMessage(type) {
    return {
        'type': type,
        'data' : []
    };
```

```
}

function getWebSocketServer(ip, port) {
    // Create HTTP server
    var server = http.createServer();
    server.listen(port, ip, function () {
        console.log("WebSocket server listening on port " + port);
    });
    // Create WebSocket server
    var wsServer = new WebSocket.Server({ server });
    return wsServer;
}

if (require.main === module) {
    run();
}
```

Frame Processor

frame-processor.js

```
const GestureSet = require('./gestures/gesture-set').GestureSet;
const GestureClass = require('./gestures/gesture-class').GestureClass;
const RingBuffer = require('ringbufferjs');

class FrameProcessor {
    constructor(config) {
        // Initialize analyzer, segmenter, datasets, recognizer and classifier
        this.analyzer = new config.analyzer.module(config.analyzer.options);
        this.segmenter = new config.segmenter.module(config.segmenter.options);
        this.gestureDataset = initDataset(config.datasets.gesture);
        this.poseDataset = initDataset(config.datasets.pose);
        if (config.general.gesture.loadOnRequest) {
            this.recognizer =  new
             ↪  config.recognizer.module(config.recognizer.options);
        } else {
            this.recognizer = new
             ↪  config.recognizer.module(config.recognizer.options,
             ↪  this.gestureDataset);
        }
        if (config.general.pose.loadOnRequest) {
            this.classifier = new
             ↪  config.classifier.module(config.classifier.options);
        } else {
            this.classifier = new
             ↪  config.classifier.module(config.classifier.options,
             ↪  this.poseDataset);
        }
        // Keep track of enabled poses and gestures
        this.enabledPoses = [];
```

```
        this.enabledGestures = [];
        // Initialize pose buffer
        this.poseBuffer = new RingBuffer(config.classifier.options.bufferLength);
        this.poseCounter = {};
        // Save config
        this.config = config;
    }

    resetContext() {
        if (this.config.general.gesture.loadOnRequest) {
            this.recognizer =  new
             ↪  this.config.recognizer.module(this.config.recognizer.options);
        }
        if (this.config.general.pose.loadOnRequest) {
            this.classifier =  new
             ↪  this.config.classifier.module(this.config.classifier.options);
        }
        this.enabledPoses = [];
        this.enabledGestures = [];
    }

    enablePose(name) {
        if (!this.enabledPoses.includes(name)) {
            // The pose is not already enabled
            if (this.config.general.pose.loadOnRequest) {
                let gestureClass = this.poseDataset.getGestureClasses().get(name);
                if (gestureClass) {
                    for (const template of gestureClass.getSamples()) {
                        this.classifier.addPose(name, template);
                    }
                } else {
                    console.error(`No pose class in the dataset with name
                     ↪  '${name}'`);
                }
            }
            this.enabledPoses.push(name);
        }
    }

    enableGesture(name) {
        if (!this.enabledGestures.includes(name)) {
            // The gesture is not already enabled
            if (this.config.general.gesture.loadOnRequest) {
                let gestureClass =
                 ↪  this.gestureDataset.getGestureClasses().get(name);
                if (gestureClass) {
                    for (const template of gestureClass.getSamples()) {
                        this.recognizer.addGesture(name, template);
                    }
                } else {
                    console.error(`No gesture class in the dataset with name
                     ↪  '${name}'`);
```

```
                }
            }
            this.enabledGestures.push(name);
        }
    }

    disablePose(name) {
        var index = this.enabledPoses.indexOf(name);
        if (index > -1) {
            // The pose was enabled, disable it
            this.enabledPoses.splice(index, 1);
            if (this.config.general.pose.loadOnRequest) {
                this.classifier.removePose(name);
            }
        }
    }

    disableGesture(name) {
        var index = this.enabledGestures.indexOf(name);
        if (index > -1) {
            // The gesture was enabled, disable it
            this.enabledGestures.splice(index, 1);
            if (this.config.general.gesture.loadOnRequest) {
                this.recognizer.removeGesture(name);
            }
        }
    }

    processFrame(frame) {
        // Get pose and data from the pose buffer
        let oldPoseInfo = "";
        let oldPoseRatio = 0;
        if (this.poseBuffer.isFull()) {
            oldPoseInfo = this.poseBuffer.deq();
            if (oldPoseInfo.pose) {
                oldPoseRatio = (this.poseCounter[oldPoseInfo.pose]--) /
                 ↪  this.config.classifier.options.bufferLength;
            }
        }
        let staticPose = "";
        try {
            staticPose = this.classifier.classify(frame).name;
        } catch(error) {
            console.error(`Classifier error: ${error}`);
        }
        let newPoseInfo = { pose: "", data: "" };
        if (staticPose && (!this.config.general.pose.sendIfRequested ||
         ↪  this.enabledPoses.includes(staticPose))) {
            newPoseInfo = { pose: staticPose, data: this.analyzer.analyze(frame) };
            if (!this.poseCounter.hasOwnProperty(staticPose)) {
                this.poseCounter[staticPose] = 1;
```

```
            } else {
                this.poseCounter[staticPose]++;
            }
        }
        this.poseBuffer.enq(newPoseInfo);
        if (oldPoseInfo.pose) {
            if (oldPoseRatio > this.config.classifier.options.poseRatioThreshold) {
                // Static pose detected
                this.segmenter.notifyRecognition();
                return { 'type': 'static', 'name': oldPoseInfo.pose, 'data':
                 ↪  oldPoseInfo.data };
            }
        } else {
            // Reset analyzer
            this.analyzer.reset();
            // Try to segment and recognize dynamic gesture
            let segment = this.segmenter.segment(frame);
            if (segment) {
                let name = "";
                try {
                    name = this.recognizer.recognize(segment).name;
                } catch (error) {
                    console.error(`Recognizer error: ${error}`);
                }
                if (name && (!this.config.general.gesture.sendIfRequested ||
                 ↪  this.enabledGestures.includes(name))) {
                    this.segmenter.notifyRecognition();
                    return { 'type': 'dynamic', 'name': name, 'data': {} };
                }
            }
        }
        // Nothing detected
        return null;
    }
}

function initDataset(config) {
    // Load the dataset
    let dataset = config.loader.loadDataset(config.name, config.directory);
    // Select/aggregate/rename classes of the dataset if required
    if (config.aggregateClasses.length != 0) {
        let newDataset = new GestureSet(dataset.name);
        config.aggregateClasses.forEach((aggregate, index) => {
            // Aggregate gesture class
            let newClass = new GestureClass(aggregate.name, index);
            let templates = [];
            // Fuse the classes into a new aggregate class
            for (const className of aggregate.classes) {
                let oldClass = dataset.getGestureClasses().get(className);
                templates = templates.concat(templates, oldClass.getSamples());
            }
```

```
            // Select a number of templates from the dataset if required
            if (config.useCustomTemplatesPerClass) {
                templates = getRandomSubarray(templates, config.templatesPerClass);
            }
            // Add the templates to the new gesture class
            for (template of templates) {
                newClass.addSample(template);
            }
            // Add the aggregate class to the new dataset
            newDataset.addGestureClass(newClass);
        });
        return newDataset
    } else {
        // Select a number of templates from the dataset if required
        if (config.useCustomTemplatesPerClass) {
            let newDataset = new GestureSet(dataset.name);
            dataset.getGestureClasses().forEach(gestureClass => {
                let newClass = new GestureClass(gestureClass.name,
                 ↪  gestureClass.index);
                let templates = gestureClass.getSamples();
                templates = getRandomSubarray(templates, config.templatesPerClass);
                // Add the templates to the new gesture class
                for (template of templates) {
                    newClass.addSample(template);
                }
                newDataset.addGestureClass(newClass);
            });
            return newDataset;
        }
        return dataset;
    }
}

//https://stackoverflow.com/questions/11935175/sampling-a-random-subset-from-an-array
function getRandomSubarray(arr, size) {
    if (size > arr.length) {
        console.log("Not enough templates!")
        return arr;
    }
    var shuffled = arr.slice(0), i = arr.length, min = i - size, temp, index;
    while (i-- > min) {
        index = Math.floor((i + 1) * Math.random());
        temp = shuffled[index];
        shuffled[index] = shuffled[i];
        shuffled[i] = temp;
    }
    return shuffled.slice(min);
}

module.exports = {
    FrameProcessor
```

```
}
```

Application Interface

app-interface.js

```
import { w3cwebsocket as W3CWebSocket } from "websocket";
import ReconnectingWebSocket from 'reconnecting-websocket';

class GestureHandler {
    constructor(addr = "ws://127.0.0.1:6442") {
        // Save callbacks for gestures, poses, and frames
        this.dynamicGestureHandlers = {};
        this.staticGestureHandler = {};
        this.forEachHandler = (gesture) => {};
        this.frameHandler = (frame) => {};
        // True if the interface is connected to the server
        this.isConnected = false;
        // The websocket client
        this.client = null;
        this.connect(addr, 10000, 3000);
    }

    /**
     * Perform an action related to a gesture.
     * @callback frameCallback
     * @param {Object} frame - The data from the sensor.
     */

    /**
     * Execute the callback each time a frame is received from the sensor.
     * @param {frameCallback} callback - The callback that handles the frame
     */
    onFrame(callback) {
        this.frameHandler = callback;
    }

    /**
     * Perform an action for each gesture.
     * @callback gesturesCallback
     * @param {string} gesture - The name of the detected gesture.
     */

    /**
     * Execute the callback each time any gesture is detected.
     * @param {gesturesCallback} callback - The callback that handles the
↪  gestures.
     */
    onEachGesture(callback) {
        this.forEachHandler = callback;
```

```
44      }
45
46      /**
47       * Perform an action related to a gesture.
48       * @callback gestureCallback
49       */
50
51      /**
52       * Execute the callback each time the gesture is detected.
53       * @param {string} gesture - The name of the gesture which should trigger the
    ↪  callback.
54       * @param {gestureCallback} callback - The callback that handles the gesture.
55       */
56      onGesture(gesture, callback) {
57          if (this.isConnected) {
58              this.client.send(JSON.stringify(getOperationMessage('addGesture',
                ↪  gesture)));
59          }
60          this.dynamicGestureHandlers[gesture] = callback;
61      }
62
63      /**
64       * Execute the callback each time the pose is detected.
65       * @param {string} pose - The name of the pose which should trigger the
    ↪  callback.
66       * @param {poseCallback} callback - The callback that handles the pose.
67       */
68      onPose(pose, callback) {
69          if (this.isConnected) {
70              this.client.send(JSON.stringify(getOperationMessage('addPose', pose)));
71          }
72          this.staticGestureHandler[pose] = callback;
73      }
74
75      /**
76       * Remove the callback associated to the gesture.
77       * @param {string} gesture - The name of the gesture for which the callback
    ↪  should be removed.
78       */
79      removeGestureHandler(gesture) {
80          if (this.dynamicGestureHandlers.hasOwnProperty(gesture)) {
81              delete this.dynamicGestureHandlers[gesture];
82              if (this.isConnected) {
83
                    ↪  this.client.send(JSON.stringify(getOperationMessage('removeGesture',
                    ↪  gesture)));
84              }
85          }
86      }
87
88      /**
```

```
89          * Remove the callback associated to the pose.
90          * @param {string} pose - The name of the pose for which the callback should be
    ↪  removed.
91          */
92         removePoseHandler(pose) {
93             if (this.staticGestureHandler.hasOwnProperty(pose)) {
94                 delete this.staticGestureHandler[pose];
95                 if (this.isConnected) {
96                     this.client.send(JSON.stringify(getOperationMessage('removePose',
                        ↪  pose)));
97                 }
98             }
99         }
100
101        /**
102         * Connect to the gesture recognizer
103         */
104        connect(addr, timeout, interval) {
105            this.client = new ReconnectingWebSocket(addr, [], {
106                constructor: W3CWebSocket,
107                connectionTimeout: timeout,  // in milliseconds
108                reconnectInterval: interval
109            });
110            this.client.onopen = function() {
111                console.log("WebSocket Client Connected");
112                this.isConnected = true;
113                // Init the message
114                let message = {
115                    'type': 'operation',
116                    'data': []
117                };
118                // Add addGestures operations to the message
119                for (const gesture of Object.keys(this.dynamicGestureHandlers)) {
120                    message.data.push({
121                        'type': 'addGesture',
122                        'name': gesture
123                    });
124                }
125                // Add addPose operations to the message
126                for (const pose of Object.keys(this.staticGestureHandler)) {
127                    message.data.push({
128                        'type': 'addPose',
129                        'name': pose
130                    });
131                }
132                // Send the message to the server
133                this.client.send(JSON.stringify(message));
134            }.bind(this);
135            // Handle messages from the server
136            this.client.onmessage = function(event) {
137                let msg = JSON.parse(event.data);
```

```
            if (msg.type === 'data') {
                for (const dataMsg of msg.data) {
                    if (dataMsg.type === 'frame') {
                        this.frameHandler(dataMsg.data);
                    } else if (dataMsg.type === 'static') {
                        if (this.staticGestureHandler.hasOwnProperty(dataMsg.name))
                        ↪ {
                            this.staticGestureHandler[dataMsg.name](dataMsg.data);
                        }
                    } else if (dataMsg.type === 'dynamic') {
                        if
                        ↪  (this.dynamicGestureHandlers.hasOwnProperty(dataMsg.name))
                        ↪ {

                            ↪  this.dynamicGestureHandlers[dataMsg.name](dataMsg.data);
                            this.forEachHandler(dataMsg.name);
                        }
                    }
                }
            }
        }.bind(this);
    }

    /**
     * Disconnect from the gesture recognizer.
     */
    disconnect() {
        this.client.close();
    }
}

function getOperationMessage(type, name) {
    return {
        'type': 'operation',
        'data': [
            {
                'type': type,
                'name': name
            }
        ]
    }
}

export default GestureHandler
```

Gestures

gesture-set.js

```
class GestureSet {
    constructor(name) {
        this.name = name;
        this.gestures = new Map();
        this.G = 0;
    }

    addGestureClass(gestureClass) {
        this.gestures.set(gestureClass.name,gestureClass);
        this.G += 1;
    }

    getGestureClasses() {
        return this.gestures;
    }

    getMinTemplate() {
        let TperG = Infinity;
        for (var gestureClass of this.gestures.values()) {
            TperG = Math.min(TperG, gestureClass.TperG);
        }
        return TperG;
    }
}

module.exports = {
    GestureSet
};
```

gesture-class.js

```
class GestureClass {
    constructor(name, index) {
        this.name = name;
        this.samples = [];
        this.index = index;
        this.TperG = 0;
    }

    addSample(sample) {
        this.samples.push(sample);
        this.TperG += 1;
    }

    getSamples() {
        return this.samples;
    }
}

module.exports = {
    GestureClass
```

```
};
```

sample.js

```
class Sample {
    constructor(user, id, infosupp) {
        this.user = user;
        this.id = id;
        this.infosupp = infosupp;
    }
}

module.exports = {
    Sample
};
```

pose-data.js

```
const Sample = require('./sample').Sample;

class PoseData extends Sample {
    constructor(user, id, frame, infosupp) {
        super(user, id, infosupp);
        this.frame = frame;
    }
}

module.exports = {
    PoseData
};
```

stroke-data.js

```
const Sample = require('./sample').Sample;

class StrokeData extends Sample {
    constructor(user, id, infosupp) {
        super(user, id, infosupp);
        this.paths = {};
    }
    addPath(label, path){
        this.paths[label] = path;
    }
}

class Stroke {
    constructor(id) {
        this.id = id;
        this.points = [];
    }
    addPoint(stroke){
```

```
        this.points.push(stroke);
    }
}

class Path {
    constructor(label) {
        this.label = label;
        this.strokes = [];
    }
    addStroke(stroke){
        this.strokes.push(stroke);
    }
}

module.exports = {
    StrokeData,
    Stroke,
    Path
};
```

point.js

```
class Point2D {
    constructor(x, y, t) {
        this.x = x;
        this.y = y;
        this.t = t;
    }
}

class Point3D {
    constructor(x, y, z, t) {
        this.x = x;
        this.y = y;
        this.z = z;
        this.t = t;
    }
}

module.exports = {
    Point2D,
    Point3D
};
```

Frames

frame.js

```
class Frame {
    constructor(timestamp) {
```

```
        this.timestamp = timestamp;
        this.hasLeftHand = false;
        this.hasRightHand = false;
        this.articulations = [];
    }

    addArticulation(articulation) {
        this.articulations.push(articulation);
    }

    getArticulation(name) {
        for (const articulation of this.articulations) {
            if (articulation.label === name) {
                return articulation;
            }
        }
        return null;
    }
}

class Articulation {
    constructor(label, point) {
        this.label = label;
        this.point = point;
    }
}

module.exports = {
    Frame,
    Articulation
}
```

Benchmarking Tool

benchmark.js

```
const path = require('path');
const fs = require('fs');
const config = require('./benchmarkConfig');
const GestureSet = require('./framework/gestures/gesture-set').GestureSet;
const GestureClass = require('./framework/gestures/gesture-class').GestureClass;

// Important values
const MINT = config.minT; // Maximum Training Templates
const MAXT = config.maxT; // Maximum Training Templates
const computeNextT = config.nextT; // Function that computes the next number of
 ↪  training templates
const R = config.r; // Repetitions
const N = config.recognizers.nSamplingPoints; // Points/Shapes

```

```
function run() {
    // // Test classifiers
    if (config.classifiers.modules.length > 0) {
        let results = [];
        for(let i=0; i < config.datasets.pose.length; i++){
            let datasetConfig = config.datasets.pose[i];
            let datasetPath = path.join(config.datasets.path, "pose");
            let dataset = loadDataset(datasetPath, datasetConfig);
            let datasetResults = {
                r: config.r,
                dataset: dataset.name,
                gestures: Array.from(dataset.getGestureClasses().keys()),
                data: []
            };
            for(let j=0; j < config.classifiers.modules.length; j++){
                let classifier = config.classifiers.modules[j];
                let res = test(dataset, classifier.module, classifier.options,
                 ↪  "classifiers");
                console.log(classifier.module.name);
                datasetResults.data.push({
                    name: classifier.module.name,
                    options: classifier.options,
                    data: res
                });
            }
            results.push(datasetResults);
        }
        fs.writeFileSync('results-poses.json', JSON.stringify(results, null, 2));
    }

    // Test recognizers
    if (config.recognizers.modules.length > 0) {
        let results = [];
        for(let i=0; i < config.datasets.gesture.length; i++){
            let datasetConfig = config.datasets.gesture[i];
            let datasetPath = path.join(config.datasets.path, "gesture");
            let dataset = loadDataset(datasetPath, datasetConfig);
            let datasetResults = {
                r: config.r,
                dataset: dataset.name,
                gestures: Array.from(dataset.getGestureClasses().keys()),
                data: []
            };
            for(let j=0; j < config.recognizers.modules.length; j++){
                let recognizer = config.recognizers.modules[j];
                let res = test(dataset, recognizer.module, recognizer.options,
                 ↪  "recognizer");
                console.log(recognizer.module.name);
                datasetResults.data.push({
                    name: recognizer.module.name,
                    options: recognizer.options,
```

```
                    data: res
                });
            }
            results.push(datasetResults);
        }
        fs.writeFileSync('results-gestures.json', JSON.stringify(results, null,
         ↪  2));
    }
}

/**
 * User-independent testing
 */
function startUserIndependentTesting(dataset, recognizerOrClassifierType, config,
 ↪  type = "recognizer") {
    let results = [];
    // Perform the test for each size of training set
    for(let trainingSetSize = MINT; trainingSetSize <=
     ↪  Math.min(dataset.getMinTemplate(), MAXT); trainingSetSize =
     ↪  computeNextT(trainingSetSize)) {
        let res = {
            n: trainingSetSize,
            accuracy: 0.0,
            time: 0.0,
            confusionMatrix: []
        };
        res.confusionMatrix = new Array(dataset.G).fill(0).map(() => new
         ↪  Array(dataset.G).fill(0));
        // Repeat the test R times
        for(let r = 0 ; r < R ; r++) {
            // Initialize the recognizer and select the candidates
            let recognizerOrClassifier = new recognizerOrClassifierType(config);
            let candidates = selectCandidates(dataset);
            // For each gesture class, mark the templates that cannot be reused
            let markedTemplates = [];
            candidates.forEach(candidate =>{
                markedTemplates.push([candidate]);
            });
            // Train the recognizer / classifier
            for(let t = 0 ; t < trainingSetSize ; t++) { // Add trainingSetSize
             ↪  strokeData per gestureClass
                // Add one sample for each gesture class
                let index = 0;
                dataset.getGestureClasses().forEach((gestureClass) => {
                    // Select a valid training template
                    let training = -1;
                    //while (training == -1 ||
                     ↪  markedTemplates[index].includes(training) ||
                     ↪  gestureClass.getSamples()[training].user ==
                     ↪  gestureClass.getSamples()[markedTemplates[index][0]].user)
                     ↪  {
```

```
                    while (training == -1 ||
                     ↪  markedTemplates[index].includes(training)) {
                        training = getRandomNumber(0,
                         ↪  gestureClass.getSamples().length);
                    }
                    // Mark the training template
                    markedTemplates[index].push(training);
                    // Train the recognizer / classifier
                    if (type === "recognizer") {
                        recognizerOrClassifier.addGesture(gestureClass.name,
                         ↪  gestureClass.getSamples()[training]);
                    } else {
                        recognizerOrClassifier.addPose(gestureClass.name,
                         ↪  gestureClass.getSamples()[training]);
                    }
                    index++;
                });
            }
            // Test the recognizer / classifier
            let index = 0;
            dataset.getGestureClasses().forEach((gestureClass) => {
                // Retrieve the testing sample
                let toBeTested = gestureClass.getSamples()[candidates[index]];
                // Attempt recognition
                if (type === "recognizer") {
                    var result = recognizerOrClassifier.recognize(toBeTested);
                } else {
                    var result = recognizerOrClassifier.classify(toBeTested.frame);
                }
                // Update the confusion matrix
                if (dataset.getGestureClasses().has(result.name)) {
                    let resultIndex =
                     ↪  dataset.getGestureClasses().get(result.name).index;
                    res.confusionMatrix[gestureClass.index][resultIndex] += 1;
                }
                // Update execution time and accuracy
                res.accuracy += (result.name===gestureClass.name) ? 1 : 0;
                res.time += result.time;
                index++;
            });
        }
        res.accuracy = res.accuracy / (R * dataset.G);
        res.time = res.time / (R * dataset.G);
        results.push(res);
    }
    return results;
};

/**
 * Return a random list of candidate gestures, 1 candidate per gesture class.
 */
```

```
function selectCandidates(dataset) {
    let candidates = [];
    dataset.getGestureClasses().forEach((value) => {
        candidates.push(getRandomNumber(0, value.getSamples().length));
    });
    return candidates;
};

/**
 * Return a random number between min and max.
 */
function getRandomNumber(min, max) {
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(Math.random()*(max - min))+min;
};

/**
 * Load a gesture dataset
 */
function loadDataset(datasetPath, datasetConfig) {
    // Load the dataset
    let dataset = datasetConfig.loader.loadDataset(datasetConfig.name,
    ↪  datasetPath);
    // Select/aggregate/rename classes of the dataset if required
    if (datasetConfig.aggregateClasses.length != 0) {
        let newDataset = new GestureSet(dataset.name);
        datasetConfig.aggregateClasses.forEach((aggregate, index) => {
            // Aggregate gesture class
            let newClass = new GestureClass(aggregate.name, index);
            let templates = [];
            // Fuse the classes into a new aggregate class
            for (const className of aggregate.classes) {
                let oldClass = dataset.getGestureClasses().get(className);
                templates = templates.concat(templates, oldClass.getSamples());
            }
            // Add the templates to the new gesture class
            for (template of templates) {
                newClass.addSample(template);
            }
            // Add the aggregate class to the new dataset
            newDataset.addGestureClass(newClass);
        });
        return newDataset
    } else {
        return dataset;
    }
}

/**
 * Test a specific recognizer/classifier with a specific dataset
```

```
 */
function test(dataset, recognizerOrClassifierType, config, type = "recognizer"){
    let result = startUserIndependentTesting(dataset, recognizerOrClassifierType,
     ↪  config, type);
    return result;
};

if (require.main === module) {
    run();
}
```

F.1.2 Hot Spots

Sensors

abstract-sensor.js

```
class AbstractSensor {
    constructor(name) {
        this.name = name;
    }

    /**
     * Process one frame from the sensor.
     * @callback frameCallback
     * @param {Object} frame - The frame to process.
     */

    /**
     * Process frames from the sensor. Executes the callback for each frame.
     * @param {frameCallback} callback - The callback that handles the frame from
 ↪  the sensor.
     * @param {Object} options - A set of options for the sensor.
     */
    loop(callback, options) {
        throw new Error('You have to implement this function');
    }

    /**
     * Stop processing frames from the sensor.
     */
    stop() {
        throw new Error('You have to implement this function');
    }

}

module.exports = {
    AbstractSensor
}
```

Datasets

unified-loader.js

```
const path = require('path');
const fs = require('fs-extra');

const GestureSet = require('../gestures/gesture-set').GestureSet;
const GestureClass = require('../gestures/gesture-class').GestureClass;
const StrokeData = require('../gestures/stroke-data').StrokeData;


function loadDataset(name, directory) {
    let gestureSet = new GestureSet(name);
    let dirPath = path.join(directory, name);
    let gestureIndex = 0;

    fs.readdirSync(dirPath).forEach((gesture) => {
        let gestureDirPath = path.join(dirPath, gesture);
        let gestureClass = new GestureClass(gesture, gestureIndex);
        gestureSet.addGestureClass(gestureClass);
        fs.readdirSync(gestureDirPath).forEach((user) => {
            let userDirPath = path.join(gestureDirPath, user);
            fs.readdirSync(userDirPath).forEach((file) => {
                let rawGesturePath = path.join(userDirPath, file);
                let strokeData = JSON.parse(fs.readFileSync(rawGesturePath));

                let filenameParsed = file.split(".")[0].split("-");
                let gestureName = filenameParsed[0];
                let infosupp;
                if(filenameParsed.length > 2){
                    infosupp = filenameParsed[1];
                }
                let id = parseInt(filenameParsed[filenameParsed.length-1]);

                let gestureData = new StrokeData(parseInt(user), id, infosupp);

                strokeData.paths.forEach(path =>{
                    gestureData.addPath(path.label, path);
                });
                gestureClass.addSample(gestureData);
            });
        });
        gestureIndex+=1;
    });

    return gestureSet;
}

function writeDataset(dataset, directory) {
    if (fs.existsSync(directory)) {
        fs.removeSync(directory);
```

```
    }
    fs.mkdirSync(directory);

    dataset.getGestureClasses().forEach(gestureClass =>{
        fs.mkdirSync(path.join(directory, gestureClass.name));
        gestureClass.samples.forEach(sample =>{
            let userDirPath = path.join(directory, gestureClass.name,
             ↪  sample.user.toString().padStart(2, "0"));
            if(!fs.existsSync(userDirPath)){
                fs.mkdirSync(userDirPath);
            }
            let newFilename = gestureClass.name + "-";
            if(sample.infosupp)
                newFilename = newFilename + sample.infosupp + "-";
            newFilename = newFilename + sample.id.toString().padStart(2, "0") +
             ↪  ".json";
            newFilename = path.join(userDirPath, newFilename);
            let toWrite = {};
            toWrite.name = gestureClass.name;
            toWrite.subject = sample.user;

            toWrite.paths = [];
            for(let path in sample.paths){
                toWrite.paths.push({
                    "label": sample.paths[path].label,
                    "strokes": sample.paths[path].strokes
                });
            }

            fs.writeFileSync(newFilename, JSON.stringify(toWrite, null, 2),
             ↪  function (err) {
                if (err) throw err;
                //console.log('Saved!');
            });
        })
    });
}

module.exports = {
    loadDataset,
    writeDataset
};
```

Classifiers

abstract-classifier.js

```
class AbstractClassifier {
    constructor(options, dataset) {
        // Empty
```

```
    }

    /**
     * Add a pose to the training set of this classifier.
     * @param {Object} name - The name of the pose to add.
     * @param {Object} frame - The frame containing the pose to add.
     */
    addPose(name, frame) {
        throw new Error('You have to implement this function');
    }

    /**
     * Remove a pose from the training set of this classifier.
     * @param {Object} name - The name of the pose to remove.
     */
    removePose(name) {
        throw new Error('You have to implement this function');
    }

    /**
     * Return the name of the pose that corresponds to the frame.
     * @param {Object} frame - A frame from the sensor.
     * @returns a JavaScript object with at least 2 properties: name (the name of
 ↪  the pose) and time (the execution time).
     */
    classify(frame) {
        throw new Error('You have to implement this function');
    }
}

module.exports = {
    AbstractClassifier
}
```

Analyzers

abstract-analyzer.js

```
class AbstractAnalyzer {
    constructor(options) {
        // Empty
    }

    /**
     * Extract and return information from a frame.
     * @param {Object} frame - A frame from the sensor.
     * @returns a JavaScript object with the information extracted from the frame.
     */
    analyze(frame) {
        throw new Error('You have to implement this function');
```

```
    }

    /**
     * Reset the internal state of the analyzer.
     */
    reset() {
        throw new Error('You have to implement this function');
    }
}

module.exports = {
    AbstractAnalyzer
}
```

Segmenters

abstract-segmenter.js

```
class AbstractSegmenter {
    constructor(options) {
        // Empty
    }

    /**
     * Process a frame and return a StrokeData object if the frame marks the end of a gesture.
     * @param {Object} frame - A frame from the sensor.
     * @returns null if the frame and the current set of frames do not correspond to a gesture, a StrokeData object otherwise.
     */
    segment(frame) {
        throw new Error('You have to implement this function');
    }

    /**
     * Indicate to the segmenter that a gesture has been recognized.
     */
    notifyRecognition() {
        throw new Error('You have to implement this function');
    }
}

module.exports = {
    AbstractSegmenter
};
```

Recognizers

abstract-recognizer.js

```
class AbstractRecognizer {
    constructor(options, dataset) {
        // Empty
    }

    /**
     * Add a gesture to the training set of this recognizer.
     * @param {Object} name - The name of the gesture to add.
     * @param {Object} sample - The gesture to add.
     */
    addGesture(name, sample) {
        throw new Error('You have to implement this function');
    }

    /**
     * Remove a gesture from the training set of this recognizer.
     * @param {Object} name - The name of the gesture to remove.
     */
    removeGesture(name) {
        throw new Error('You have to implement this function');
    }

    /**
     * Check whether the sample corresponds to a known gesture.
     * @param {Object} sample - A gesture sample from the sensor.
     * @returns a JavaScript object with at least 2 properties: name (the name of
 ↪  the gesture) and time (the execution time).
     */
    recognize(sample) {
        throw new Error('You have to implement this function');
    }
}

module.exports = {
    AbstractRecognizer
};
```

F.1.3 Configuration Files

Framework Configuration

config.js

```
// MODULES ---------------------------------------------------------------------
// Sensor Interfaces
const LeapSensor = require('./implementation/sensors/leap-sensor/sensor').Sensor;

// Pose Analyzers
const NoAnalyzer =
 ↪  require('./implementation/analyzers/placeholder-analyzer/analyzer').Analyzer;
```

```
const BasicAnalyzer =
    require('./implementation/analyzers/basic-analyzer/analyzer').Analyzer;

// Gesture Segmenters
const WindowSegmenter =
    require('./implementation/segmenters/window-segmenter').Segmenter;
const ZoningSegmenter =
    require('./implementation/segmenters/zoning-segmenter').Segmenter;
const LeftHandSegmenter =
    require('./implementation/segmenters/lefthand-segmenter').Segmenter;
const FrameSegmenter =
    require('./implementation/segmenters/frame-segmenter').Segmenter;

// Gesture Recognizers
const NoRecognizer =
    require('./implementation/recognizers/placeholder-recognizer/recognizer').Recognizer;
//3D
const HybridP3DollarPlusXRecognizer =
    require('./implementation/recognizers/hybridp3dollarplusx-recognizer/recognizer').Recogni
const JackknifeRecognizer =
    require('./implementation/recognizers/jackknife-recognizer/recognizer').Recognizer;
const P3DollarPlusXRecognizer =
    require('./implementation/recognizers/p3dollarplusx-recognizer/recognizer').Recognizer;
const UVPRecognizer =
    require('./implementation/recognizers/uvplusflexiblecloud-recognizer/recognizer').Recogni
const ThreeCentRecognizer =
    require('./implementation/recognizers/threecent-recognizer/recognizer').Recognizer;
const P3DollarPlusRecognizer =
    require('./implementation/recognizers/p3dollarplus-recognizer/recognizer').Recognizer;
//2D
const PDollarPlusRecognizer =
    require('./implementation/recognizers/pdollarplus-recognizer/recognizer').Recognizer;

// Pose Classifiers
const NoClassifier =
    require('./implementation/classifiers/placeholder-classifier/classifier').Classifier;
const GPSDClassifier =
    require('./implementation/classifiers/gpsd-classifier/classifier').Classifier;
const GPSDaClassifier =
    require('./implementation/classifiers/gpsda-classifier/classifier').Classifier;
const GPSDaDissimilarityClassifier =
    require('./implementation/classifiers/gpsdadissimilarity-classifier/classifier').Classifi
const P3DollarPlusClassifier =
    require('./implementation/classifiers/p3dollarplus-classifier/classifier').Classifier;

// Gesture Dataset Loaders
const UnifiedDatasetLoader = require('./framework/datasets/unified-loader');
const GuinevereDatasetLoader =
    require('./implementation/datasets/gesture/guinevere-loader');
const LeapMotionDatasetLoader =
    require('./implementation/datasets/gesture/leap-gesture-loader');
```

```
38 const SmartphoneDatasetLoader =
    ↪ require('./implementation/datasets/gesture/smartphone-loader');
39 const HandGestureDatasetLoader =
    ↪ require('./implementation/datasets/gesture/hand-gesture-csv-loader');
40 const UWaveDatasetLoader =
    ↪ require('./implementation/datasets/gesture/uwave-loader');
41
42 // Pose Dataset Loaders
43 const MMHGRDatasetLoader = require('./implementation/datasets/pose/mmhgr-loader');
44 const LeapPoseDatasetLoader =
    ↪ require('./implementation/datasets/pose/leap-pose-loader');
45
46 // Articulations
47 const fingerMcps = {
48     left: ["leftThumbMcpPosition", "leftIndexMcpPosition", "leftMiddleMcpPosition",
        ↪ "leftRingMcpPosition", "leftPinkyMcpPosition"],
49     right: ["rightThumbMcpPosition", "rightIndexMcpPosition",
        ↪ "rightMiddleMcpPosition", "rightRingMcpPosition", "rightPinkyMcpPosition"]
50 }
51 const fingerPips = {
52     left: ["leftThumbPipPosition", "leftIndexPipPosition", "leftMiddlePipPosition",
        ↪ "leftRingPipPosition", "leftPinkyPipPosition"],
53     right: ["rightThumbPipPosition", "rightIndexPipPosition",
        ↪ "rightMiddlePipPosition", "rightRingPipPosition", "rightPinkyPipPosition"]
54 }
55 const fingerTips = {
56     left: ["leftThumbTipPosition", "leftIndexTipPosition", "leftMiddleTipPosition",
        ↪ "leftRingTipPosition", "leftPinkyTipPosition"],
57     right: ["rightThumbTipPosition", "rightIndexTipPosition",
        ↪ "rightMiddleTipPosition", "rightRingTipPosition", "rightPinkyTipPosition"]
58 }
59 const palms = {
60     left: ["leftPalmPosition"],
61     right: ["rightPalmPosition"]
62 }
63 const articulationsBothHandsSimple = [].concat(palms.right, palms.left,
    ↪ fingerTips.right, fingerTips.left);
64 const articulationsRightHandSimple = [].concat(palms.right, fingerTips.right);
65 const articulationsRightHandDetailed = [].concat(palms.right, fingerMcps.right,
    ↪ fingerPips.right, fingerTips.right);
66
67 // CONFIG INIT ------------------------------------------------------------------
68 var config = {};
69 config.general = {};
70 config.general.pose = {};
71 config.general.gesture = {};
72 config.server = {};
73 config.datasets = {};
74
75 // CONFIGURATION ----------------------------------------------------------------
76 // General Configuration
```

```
config.general.debug = false;                          // Show debug logs
config.general.sendContinuousData = true;              // Send data from each frame to
 ↪  the client
config.general.gesture = {
    sendIfRequested: true,                             // Send recognized gestures
     ↪  only if they are requested by the client
    loadOnRequest: false                               // Load gestures based on
     ↪  requests from the client
}
config.general.pose = {
    sendIfRequested: true,                             // Send recognized gestures
     ↪  only if they are requested by the client
    loadOnRequest: false                               // Load gestures based on
     ↪  requests from the client
}

// Server
config.server.ip = '127.0.0.1';         // IP of the server (for app interface)
config.server.port = 6442;             // Port of the server (for app interface)

// Sensor Interface
config.sensor = {
    module: LeapSensor,
    options: {
        framerate: 60           // Sensor framerate [images/seconds]
    }
}

// Pose Analyzer
config.analyzer = {
    module: BasicAnalyzer,
    options: {}
}

// Gesture Segmenter
config.segmenter = {
    module: WindowSegmenter,
    options: {
        motionThreshold: 90,                           // Minimum hand motion to
         ↪  consider that the segment may be a gesture
        minSegmentLength: 10,                          // Minimum length of a segment
         ↪  (if applicable) [#frames]
        maxSegmentLength: 60,                          // Maximum length of a segment
         ↪  (if applicable) [#frames]
        windowWidth: 20,                               // Width of the window (if
         ↪  applicable) [#frames]
        intervalLength: 3,                             // Length of the interval
         ↪  between 2 consecutive segments (if applicable) [#frames]
        pauseLength: 60,                               // Length of the pause after a
         ↪  gesture has been detected (if applicable) [#frames]
        xBound: 120,                                   // 1/2 width of the zone (if
         ↪  applicable) [mm]
```

```
        zBound: 60                                  // 1/2 depth of the zone (if
        ↪ applicable) [mm]
    }
}

// Gesture Dataset
config.datasets.gesture = {
    directory: __dirname + "/datasets/gesture",
    loader: LeapMotionDatasetLoader,
    name: "guinevere",
    useCustomTemplatesPerClass: true,
    templatesPerClass: 16,
    aggregateClasses: [
        { name: "rhand_uswipe", classes: ["swipe_up"] },
        { name: "rhand_dswipe", classes: ["swipe_down"] },
        { name: "rhand_lswipe", classes: ["swipe_left"] },
        { name: "rhand_rswipe", classes: ["swipe_right"] },
        { name: "rindex_airtap", classes: ["tap"] },
        //{ name: "rhand_crotate", classes: ["turn_clockwise"] },
        //{ name: "rhand_acrotate", classes: ["turn_counter_clockwise"] },
        //{ name: "rhand_close", classes: ["pinch_in"] },
        //{ name: "rhand_open", classes: ["pinch_out"] },
        //{ name: "thumbs_up", classes: ["thumbs_up"] },
        //{ name: "thumbs_down", classes: ["thumbs_down"] }
    ]
}

// Pose Dataset
config.datasets.pose = {
    directory: __dirname + "/datasets/pose",
    loader: LeapPoseDatasetLoader,
    name: "guinevre-pose",
    useCustomTemplatesPerClass: true,
    templatesPerClass: 100,
    aggregateClasses: []
}

// Gesture Recognizer
config.recognizer = {
    module: JackknifeRecognizer,
    options: {
        palmThreshold: 50,
        fingerThreshold: 15,
        samplingPoints: 16,                         // Number of sampling points
        ↪ [#points]
        articulations: palms.right,
        //pathName: "rightPalmPosition"
    }
}

// Pose Classifier
```

```
config.classifier = {
    module: GPSDaClassifier,
    options: {
        bufferLength: 15,
        poseRatioThreshold: 0.8,
        alpha: 0.8,
        articulations: articulationsRightHandDetailed
    }
}

module.exports = config;
```

Benchmark Configuration

benchmarkConfig.js

```
// MODULES
 ↪  --------------------------------------------------------------------------
const path = require('path');

// Gesture datasets loaders
const UnifiedDatasetLoader = require('./framework/datasets/unified-loader');
const GuinevereDatasetLoader =
 ↪  require('./implementation/datasets/gesture/guinevere-loader');
const LeapMotionDatasetLoader =
 ↪  require('./implementation/datasets/gesture/leap-gesture-loader');
const SmartphoneDatasetLoader =
 ↪  require('./implementation/datasets/gesture/smartphone-loader');
const HandGestureDatasetLoader =
 ↪  require('./implementation/datasets/gesture/hand-gesture-csv-loader');
const UWaveDatasetLoader =
 ↪  require('./implementation/datasets/gesture/uwave-loader');

// Pose datasets loaders
const MMHGRDatasetLoader = require('./implementation/datasets/pose/mmhgr-loader');
const LeapPoseDatasetLoader =
 ↪  require('./implementation/datasets/pose/leap-pose-loader');

// Gesture recognizers
//3D
const HybridP3DollarPlusXRecognizer = require('./implementation/
 ↪  recognizers/hybridp3dollarplusx-recognizer/recognizer').Recognizer;
const JackknifeRecognizer = require('./implementation/
 ↪  recognizers/jackknife-recognizer/recognizer').Recognizer;
const P3DollarPlusXRecognizer = require('./implementation/
 ↪  recognizers/p3dollarplusx-recognizer/recognizer').Recognizer;
const UVPRecognizer = require('./implementation/
 ↪  recognizers/uvplusflexiblecloud-recognizer/recognizer').Recognizer;
const ThreeCentRecognizer = require('./implementation/
 ↪  recognizers/threecent-recognizer/recognizer').Recognizer;
```

```
23 const P3DollarRecognizer = require('./implementation/
    ↪ recognizers/p3dollar-recognizer/recognizer').Recognizer;
24 const P3DollarPlusRecognizer = require('./implementation/
    ↪ recognizers/p3dollarplus-recognizer/recognizer').Recognizer;
25 const Q3DollarRecognizer = require('./implementation/
    ↪ recognizers/q3dollar-recognizer/recognizer').Recognizer;
26 //2D
27 const PDollarPlusRecognizer = require('./implementation/
    ↪ recognizers/pdollarplus-recognizer/recognizer').Recognizer;
28
29 // Pose classifiers
30 const GPSDClassifier = require('./implementation/
    ↪ classifiers/gpsd-classifier/classifier').Classifier;
31 const GPSDaClassifier = require('./implementation/
    ↪ classifiers/gpsda-classifier/classifier').Classifier;
32 const GPSDaDissimilarityClassifier = require('./implementation/
    ↪ classifiers/gpsdadissimilarity-classifier/classifier').Classifier;
33 const P3DollarPlusClassifier = require('./implementation/
    ↪ classifiers/p3dollarplus-classifier/classifier').Classifier;
34
35 // Articulations
36 const fingerMcps = {
37     left: ["leftThumbMcpPosition", "leftIndexMcpPosition", "leftMiddleMcpPosition",
        ↪ "leftRingMcpPosition", "leftPinkyMcpPosition"],
38     right: ["rightThumbMcpPosition", "rightIndexMcpPosition",
        ↪ "rightMiddleMcpPosition", "rightRingMcpPosition", "rightPinkyMcpPosition"]
39 }
40 const fingerPips = {
41     left: ["leftThumbPipPosition", "leftIndexPipPosition", "leftMiddlePipPosition",
        ↪ "leftRingPipPosition", "leftPinkyPipPosition"],
42     right: ["rightThumbPipPosition", "rightIndexPipPosition",
        ↪ "rightMiddlePipPosition", "rightRingPipPosition", "rightPinkyPipPosition"]
43 }
44 const fingerTips = {
45     left: ["leftThumbTipPosition", "leftIndexTipPosition", "leftMiddleTipPosition",
        ↪ "leftRingTipPosition", "leftPinkyTipPosition"],
46     right: ["rightThumbTipPosition", "rightIndexTipPosition",
        ↪ "rightMiddleTipPosition", "rightRingTipPosition", "rightPinkyTipPosition"]
47 }
48 const palms = {
49     left: ["leftPalmPosition"],
50     right: ["rightPalmPosition"]
51 }
52 const articulationsBothHandsSimple = [].concat(palms.right, palms.left,
    ↪ fingerTips.right, fingerTips.left);
53 const articulationsRightHandSimple = [].concat(palms.right, fingerTips.right);
54 const articulationsRightHandDetailed = [].concat(palms.right, fingerMcps.right,
    ↪ fingerPips.right, fingerTips.right);
55
56
57 // CONFIG INIT
    ↪ ------------------------------------------------------------------
```

```
var config = {};
config.datasets = {};
config.recognizers = {};
config.classifiers = {};

// CONFIGURATION ----------------------------------------------------------------
// General configuration
config.minT = 1; // Minimum Training Templates
config.maxT = 8; // Maximum Training Templates (included)
config.nextT = x => x * 2; // Function used to increment the number of training templates
config.r = 100; // Repetitions

// Recognizers
N_SAMPLING_POINTS = 16;
config.recognizers.nSamplingPoints = N_SAMPLING_POINTS;
config.recognizers.modules = [
    {
        module: UVPRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            articulations: articulationsRightHandSimple
        }
    },
    {
        module: UVPRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            articulations: ["rightPalmPosition"]
        }
    },
    {
        module: JackknifeRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            articulations: articulationsRightHandSimple
        }
    },
    {
        module: JackknifeRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            articulations: ["rightPalmPosition"]
        }
    },
    {
        module: ThreeCentRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            pathName: "rightPalmPosition"
```

```
        }
    },
    {
        module: P3DollarRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            articulations: articulationsRightHandSimple
        }
    },
    {
        module: P3DollarRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            articulations: ["rightPalmPosition"]
        }
    },
    {
        module: P3DollarPlusRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            articulations: articulationsRightHandSimple
        }
    },
    {
        module: P3DollarPlusRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            articulations: ["rightPalmPosition"]
        }
    },
    {
        module: Q3DollarRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            articulations: articulationsRightHandSimple
        }
    },
    {
        module: Q3DollarRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            articulations: ["rightPalmPosition"]
        }
    },
    {
        module: P3DollarPlusXRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            articulations: articulationsRightHandSimple
        }
    },
```

```
    {
        module: P3DollarPlusXRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            articulations: ["rightPalmPosition"]
        }
    },
    {
        module: HybridP3DollarPlusXRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            palmThreshold: 50,
            fingerThreshold: 0
        }
    },

    {
        module: HybridP3DollarPlusXRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            palmThreshold: 50,
            fingerThreshold: 5
        }
    },
    {
        module: HybridP3DollarPlusXRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            palmThreshold: 1000,
            fingerThreshold: 0
        }
    },
    {
        module: PDollarPlusRecognizer,
        options: {
            samplingPoints: N_SAMPLING_POINTS,
            pathName: "rightPalmPosition"
        }
    }
];

// Classifiers
config.classifiers.modules = [
    {
        module: GPSDaClassifier,
        options: {
            alpha: 0.60,
            articulations: articulationsRightHandSimple
        }
    },
    {
```

```
        module: GPSDaClassifier,
        options: {
            alpha: 0.60,
            articulations: articulationsRightHandDetailed
        }
    },
    {
        module: GPSDaClassifier,
        options: {
            alpha: 0.70,
            articulations: articulationsRightHandSimple
        }
    },
    {
        module: GPSDaClassifier,
        options: {
            alpha: 0.70,
            articulations: articulationsRightHandDetailed
        }
    },
    {
        module: GPSDaClassifier,
        options: {
            alpha: 0.80,
            articulations: articulationsRightHandSimple
        }
    },
    {
        module: GPSDaClassifier,
        options: {
            alpha: 0.80,
            articulations: articulationsRightHandDetailed
        }
    },
    {
        module: GPSDaClassifier,
        options: {
            alpha: 0.90,
            articulations: articulationsRightHandSimple
        }
    },
    {
        module: GPSDaClassifier,
        options: {
            alpha: 0.90,
            articulations: articulationsRightHandDetailed
        }
    },
    {
        module: GPSDaClassifier,
        options: {
```

```
            alpha: 1.00,
            articulations: articulationsRightHandSimple
        }
    },
    {
        module: GPSDaClassifier,
        options: {
            alpha: 1.00,
            articulations: articulationsRightHandDetailed
        }
    },
    {
        module: GPSDaClassifier,
        options: {
            alpha: 1.10,
            articulations: articulationsRightHandSimple
        }
    },
    {
        module: GPSDaClassifier,
        options: {
            alpha: 1.10,
            articulations: articulationsRightHandDetailed
        }
    },
    {
        module: GPSDaDissimilarityClassifier,
        options: {
            alpha: 0.7,
            articulations: articulationsRightHandSimple
        }
    },
    {
        module: GPSDaDissimilarityClassifier,
        options: {
            alpha: 0.7,
            articulations: articulationsRightHandDetailed
        }
    },
    {
        module: P3DollarPlusClassifier,
        options: {
            articulations: articulationsRightHandSimple
        }
    },
    {
        module: P3DollarPlusClassifier,
        options: {
            articulations: articulationsRightHandDetailed
        }
    }
```

```
];

// Datasets path
config.datasets.path = path.join(__dirname, "datasets");

// Gesture datasets
config.datasets.gesture = [
    {
        name: "guinevere_unified",
        loader: UnifiedDatasetLoader,
        aggregateClasses: [
            { name: "swipe up", classes: ["swipe_up"] },
            { name: "swipe down", classes: ["swipe_down"] },
            { name: "swipe left", classes: ["swipe_left"] },
            { name: "swipe right", classes: ["swipe_right"] },
            { name: "tap", classes: ["tap"] },
            { name: "rotate clockwise", classes: ["turn_clockwise"] },
            { name: "rotate anti-clockwise", classes: ["turn_counter_clockwise"] },
            { name: "pinch in", classes: ["pinch_in"] },
            { name: "pinch out", classes: ["pinch_out"] },
            { name: "thumbs up", classes: ["thumbs_up"] },
            { name: "thumbs down", classes: ["thumbs_down"] }
        ]
    },
    {
        name: "guinevere_unified",
        loader: UnifiedDatasetLoader,
        aggregateClasses: [
            { name: "swipe up", classes: ["swipe_up"] },
            { name: "swipe left", classes: ["swipe_left"] },
            { name: "swipe right", classes: ["swipe_right"] },
            { name: "tap", classes: ["tap"] }
        ]
    }
    // {
    //     name: "leapmotion",
    //     loader: LeapMotionDatasetLoader,
    //     aggregateClasses: []
    // }
    // {
    //     name: "uWaveGestureLibrary",
    //     loader: UWaveDatasetLoader,
    //     aggregateClasses: []
    // }
    // {
    //     name: "smartphone",
    //     loader: SmartphoneDatasetLoader,
    //     aggregateClasses: []
    // }
];

```

```
// Pose datasets
config.datasets.pose = [
    {
        loader: LeapPoseDatasetLoader,
        name: "guinevre-pose",
        aggregateClasses: [
            { name: "flat", classes: ["flat"] },
            { name: "point", classes: ["point-index"] },
            { name: "pinch", classes: ["pinch"] },
            { name: "grab", classes: ["grab"] },
            { name: "thumb", classes: ["thumb"] }
        ]
    }
    // {
    //     loader: MMHGRDatasetLoader,
    //     name: "multi_mod_hand_gest_recog"
    //     aggregateClasses: []
    // }
];

module.exports = config;
```

F.2 Sensors

F.2.1 Leap Motion Controller

sensor.js

```
const AbstractSensor =
 ↪  require('../../../framework/sensors/abstract-sensor').AbstractSensor
const Point = require('../../../framework/gestures/point').Point3D;
const { Frame, Articulation } = require('../../../framework/frames/frame');
const Leap = require('leapjs');

const fingerNames = ["Thumb", "Index", "Middle", "Ring", "Pinky"];
const fingerArticulations = ["Mcp", "Pip", "Tip"];

class Sensor extends AbstractSensor {
    constructor(options) {
        super("Leap-Interface");
        this.framerate = options.framerate;
        this.controller = new Leap.Controller({
            frameEventName: 'deviceFrame',
            loopWhileDisconnected: false
        })
        this.sensorLoop = null;
    }

    loop(callback) {
        this.controller.connect()
        this.callback = callback;
        let hadRightHand = false;
        let processLeapFrame = function () {
            let frame = this.controller.frame();
            let parsedFrame = new Frame(frame.timestamp);
            let fingers = [];
            // Palm positions
            for (const hand of frame.hands) {
                let palmName;
                if (hand.type === 'right') {
                    palmName = "rightPalmPosition";
                    hadRightHand = true;
                    parsedFrame.hasRightHand = true;
                }
                else {
                    palmName = "leftPalmPosition";
                    parsedFrame.hasLeftHand = true;
                }
                let palmPosition = hand.palmPosition;
                parsedFrame.articulations.push({ label: palmName, point: new
                 ↪  Point(palmPosition[0], palmPosition[1], palmPosition[2],
                 ↪  frame.timestamp) });
                hand.fingers.forEach((finger) => {
```

```
                    if (hand.type === 'right') {
                        // Get data usable by the application
                        let position = finger.stabilizedTipPosition;
                        let normalized =
                         ↪  frame.interactionBox.normalizePoint(position);
                        fingers.push({
                            'type': finger.type,
                            'normalizedPosition': normalized,
                            'touchDistance': finger.touchDistance,
                            'tipVelocity': finger.tipVelocity
                        });
                    }
                    for (const fingerArticulation of fingerArticulations) {
                        // Get label (e.g., rightIndexPipPosition)
                        let label = `${hand.type}${fingerNames[finger.type]}
                         ↪  ${fingerArticulation}Position`;
                        let position = new Point(
                         ↪  ...finger[`${fingerArticulation.toLowerCase()}Position`]);
                        let articulation = new Articulation(label, position);
                        parsedFrame.addArticulation(articulation);
                    }
                });
            }
            // Add points if articulation not visible
            if (!parsedFrame.hasRightHand) {
                let label = `rightPalmPosition`;
                let articulation = new Articulation(label, new Point(0.0, 0.0, 0.0,
                 ↪  frame.timestamp));
                parsedFrame.addArticulation(articulation);
                for (const fingerName of fingerNames) {
                    for (const fingerArticulation of fingerArticulations) {
                        let label =
                         ↪  `right${fingerName}${fingerArticulation}Position`;
                        let articulation = new Articulation(label, new Point(0.0,
                         ↪  0.0, 0.0, frame.timestamp));
                        parsedFrame.addArticulation(articulation);
                    }
                }
            }
            if (!parsedFrame.hasLeftHand) {
                let label = `leftPalmPosition`;
                let articulation = new Articulation(label, new Point(0.0, 0.0, 0.0,
                 ↪  frame.timestamp));
                parsedFrame.addArticulation(articulation);
                for (const fingerName of fingerNames) {
                    for (const fingerArticulation of fingerArticulations) {
                        let label =
                         ↪  `left${fingerName}${fingerArticulation}Position`;
                        let articulation = new Articulation(label, new Point(0.0,
                         ↪  0.0, 0.0, frame.timestamp));
                        parsedFrame.addArticulation(articulation);
```

```
                    }
                }
            }
            let appData = { 'fingers': fingers };
            // Callback only if a hand is visible
            if (parsedFrame.hasRightHand || parsedFrame.hasLeftHand) {
                callback(parsedFrame, appData);
            } else if (fingers.length == 0 && hadRightHand) {
                callback(parsedFrame, appData);
                hadRightHand = false;
            }
        }.bind(this);

        this.sensorLoop = setInterval(processLeapFrame, 1000/this.framerate);
    }

    stop() {
        if (this.sensorLoop !== null) {
            clearInterval(this.sensorLoop);
            this.controller.disconnect();
            this.sensorLoop = null;
        }
    }
}

module.exports = {
    Sensor
}
```

F.3 Datasets

F.3.1 Static Gestures

Custom Leap Motion Pose Dataset

leap-pose-loader.js

```
const path = require('path');
const fs = require('fs');

const GestureSet = require('../../../framework/gestures/gesture-set').GestureSet;
const GestureClass =
 ↪  require('../../../framework/gestures/gesture-class').GestureClass;
const PoseData = require('../../../framework/gestures/pose-data').PoseData;
const { Frame, Articulation } = require('../../../framework/frames/frame');
const Point = require('../../../framework/gestures/point').Point3D;

const fingerNames = ["Thumb", "Index", "Middle", "Ring", "Pinky"];
const fingerArticulations = ["Mcp", "Pip", "Tip"];

function loadDataset(name, directory) {
    let gestureSet = new GestureSet(name);
    let dirPath = path.join(directory, name);
    let gestureIndex = 0;
    fs.readdirSync(dirPath).forEach((gesture) => {
        let gestureDirPath = path.join(dirPath, gesture);
        let gestureClass = new GestureClass(gesture, gestureIndex);
        gestureSet.addGestureClass(gestureClass);
        fs.readdirSync(gestureDirPath).forEach((user) => {
            let userDirPath = path.join(gestureDirPath, user);
            fs.readdirSync(userDirPath).forEach((file) => {
                let rawGesturePath = path.join(userDirPath, file);
                let parsedFile = JSON.parse(fs.readFileSync(rawGesturePath));
                let id = 0;
                for (const frame of parsedFile.data) {
                    let parsedFrame = parseFrame(frame);
                    let poseData = new PoseData(parseInt(user), id, parsedFrame,
                     ↪  undefined);
                    gestureClass.addSample(poseData);
                    id++;
                }
            });
        });
        gestureIndex += 1;
    });
    return gestureSet;
}

function parseFrame(frame) {
    let parsedFrame = new Frame(frame.id);
```

```
    let rightHandId = -1;
    // Add palms
    for (const hand of frame.hands) {
        // Save id of right hand
        if (hand.type === "right") {
            rightHandId = hand.id;
        }
        let label = `${hand.type}PalmPosition`;
        let position = new Point(...hand.palmPosition);
        let articulation = new Articulation(label, position);
        parsedFrame.addArticulation(articulation);
    }
    // Add fingers
    for (const pointable of frame.pointables) {
        if (!pointable.tool) {
            for (const fingerArticulation of fingerArticulations) {
                // Get label (e.g., rightIndexPipPosition)
                let side = pointable.handId == rightHandId ? "right" : "left";
                let label =
                 ↪  `${side}${fingerNames[pointable.type]}${fingerArticulation}
                 ↪  Position`;
                let position = new
                 ↪  Point(...pointable[`${fingerArticulation.toLowerCase()}
                 ↪  Position`]);
                let articulation = new Articulation(label, position);
                parsedFrame.addArticulation(articulation);
            }
        }
    }
    return parsedFrame;
}

module.exports = {
    loadDataset
}
```

MultiModHandGestRecog Dataset

mmhgr-loader.js

```
const path = require('path');
const fs = require('fs');
const xml2js = require('xml2js');

const { Frame, Articulation } = require('../../../framework/frames/frame');
const GestureClass =
 ↪  require('../../../framework/gestures/gesture-class').GestureClass;
const GestureSet = require('../../../framework/gestures/gesture-set').GestureSet;
const PoseData = require('../../../framework/gestures/pose-data').PoseData;
const Point = require('../../../framework/gestures/point').Point3D;
```

```
10
11  const fingers = ["Thumb", "Index", "Middle", "Ring", "Pinky"];
12  //const classNames = ["C", "down", "fist_moved", "five", "four", "hang", "heavy",
    ↪  "index", "L", "ok", "palm", "palm_m", "palm_u", "three", "two", "up"];
13  const classNames = ["C", "down"];
14
15  function loadDataset(name, directory) {
16      let datasetPath = path.join(directory, name);
17      console.log("Loading dataset (this might take some time)...");
18      let dataset = parseGestureSet(datasetPath, "MultiModHandGestRecog");
19      console.log("Data set loaded!");
20      return dataset;
21  }
22
23  // Helper functions
24  function parseFrame(path) {
25      let xml = fs.readFileSync(path);
26      let parsedFrame;
27      // Parse the xml file
28      var parser = new xml2js.Parser({explicitArray: false, mergeAttrs: true});
29      parser.parseString(xml, (err, res) => {
30          // Extract the frame
31          let frame = res.opencv_storage.Frame;
32          // Initialize the parsed frame
33          parsedFrame = new Frame(frame.ID);
34          for (const hand of ["Left", "Right"]) {
35              if (frame.Images.LeftImage.Hands.hasOwnProperty(hand)) {
36                  // Set hasXHand property
37                  if (hand === "Right") {
38                      parsedFrame.hasRightHand = true;
39                  } else {
40                      parsedFrame.hasLeftHand = true;
41                  }
42                  // Extract the position of the palm [x, y, z]
43                  let palmPosition =
                     ↪  frame.Images.LeftImage.Hands[hand].Center.data.split(/\s+/g).
                     ↪  slice(1).map(str => parseFloat(str));
44                  parsedFrame.addArticulation(new
                     ↪  Articulation(getArticulationLabel(hand === "Right", "Palm"),
                     ↪  new Point(...palmPosition)));
45                  // Extract the position of each finger
46                  let frameFingers = frame.Images.LeftImage.Hands[hand].Fingers;
47                  for (const finger of fingers) {
48                      // [x, y, z]
49                      let fingerPosition =
                         ↪  frameFingers[finger].TipPosition.data.split(/\s+/g).
                         ↪  slice(1).map(str => parseFloat(str));
50                      parsedFrame.addArticulation(new
                         ↪  Articulation(getArticulationLabel(hand === "Right", finger
                         ↪  + "Tip"), new Point(...fingerPosition)));
51                  }
```

```
            }
        }
    })
    return parsedFrame;
}

function parseGestureSet(datasetPath, name) {
    let gestureSet = new GestureSet(name);
    let classIndex = 0;
    for (const className of classNames) {
        console.log(className)
        let gestureClass = new GestureClass(className, classIndex);
        classIndex += 1;
        fs.readdirSync(datasetPath).forEach((userDir) => {
            for (const type of ["train_pose", "test_pose"]) {
                let classPath = path.join(datasetPath, userDir, type, className);
                let id = 0;
                fs.readdirSync(classPath).forEach((frame) => {
                    let framePath = path.join(classPath, frame);
                    let parsedFrame = parseFrame(framePath);
                    let userId = parseInt(userDir.split("_"));
                    let poseData = new PoseData(userId, id, parsedFrame,
                     ↪  undefined);
                    gestureClass.addSample(poseData);
                    id++;
                });
            }
        });
        gestureSet.addGestureClass(gestureClass);
    }
    return gestureSet;
}

function getArticulationLabel(isRight, name) {
    return `${isRight ? "right" : "left"}${name}Position`;
}

module.exports = {
    loadDataset
}
```

F.3.2 Dynamic Gestures

Custom Leap Motion Gesture Dataset

leap-gesture-loader.js

```
const fs = require('fs');
const path = require('path');

```

```
const GestureSet = require('../../../framework/gestures/gesture-set').GestureSet;
const GestureClass =
 ↪  require('../../../framework/gestures/gesture-class').GestureClass;
const StrokeData = require('../../../framework/gestures/stroke-data').StrokeData;
const Stroke = require('../../../framework/gestures/stroke-data').Stroke;
const Path = require('../../../framework/gestures/stroke-data').Path;
const Point = require('../../../framework/gestures/point').Point3D;

const fingerNames = ["Thumb", "Index", "Middle", "Ring", "Pinky"];
const fingerArticulations = ["Mcp", "Pip", "Tip"];

const device = {
    'osBrowserInfo': 'Leap Motion Controller', 'resolutionHeight': null,
    'resolutionWidth': null, 'windowHeight': null, 'windowWidth': null,
    'pixelRatio': null, 'mouse': false, 'pen': false, 'finger': false,
    'acceleration': false, 'webcam': true
};

function loadDataset(name, directory) {
    let gestureSet = new GestureSet(name);
    let dirPath = path.join(directory, name);
    let gestureIndex = 0;
    fs.readdirSync(dirPath).forEach((user_dir) => {
        let gestureSampleDirPath = path.join(dirPath, user_dir);
        fs.readdirSync(gestureSampleDirPath).forEach((sample) => {
            let rawGesturePath = path.join(gestureSampleDirPath, sample);
            let rawGestureData = JSON.parse(fs.readFileSync(rawGesturePath));
            let filename = sample.split(".")[0].split("-");
            let gestureName = filename[0].split("#")[0];
            let infosupp = undefined;
            if (filename[0].split("#").length > 1) {
                infosupp = filename[0].split("#")[1];
            }
            let id = filename[1];
            let gestureData = new StrokeData(parseInt(user_dir), id, infosupp);
            if (gestureSet.getGestureClasses().has(gestureName)) {
                gestureSet.getGestureClasses().get(gestureName).addSample(
                 ↪  gestureData);
            } else {
                let gestureClass = new GestureClass(gestureName, gestureIndex);
                gestureIndex += 1;
                gestureClass.addSample(gestureData);
                gestureSet.addGestureClass(gestureClass);
            }
            // Init stroke paths
            for (const side of ["left", "right"]) {
                let label = `${side}PalmPosition`
                let strokePath = new Path(label);
                gestureData.addPath(label, strokePath);
                let stroke = new Stroke(0);
                strokePath.addStroke(stroke);
```

```
                for (const fingerName of fingerNames) {
                    for (const fingerArticulation of fingerArticulations) {
                        let label =
                        ↪  `${side}${fingerName}${fingerArticulation}Position`;
                        let strokePath = new Path(label);
                        gestureData.addPath(label, strokePath);
                        let stroke = new Stroke(0);
                        strokePath.addStroke(stroke);
                    }
                }
            }
            for (let i = 0; i < rawGestureData['data'].length; i++) {
                let frame = rawGestureData['data'][i];
                let points = {};
                for (const side of ["left", "right"]) {
                    points[`${side}PalmPosition`] = null;
                    for (const fingerName of fingerNames) {
                        for (const fingerArticulation of fingerArticulations) {
                            let label =
                            ↪  `${side}${fingerName}${fingerArticulation}Position`;
                            points[label] = null;
                        }
                    }
                }
                let rightHandId = -1;
                let leftHandId = -1;
                for (const hand of frame['hands']) {
                    let palmName;
                    if (hand['type'] === 'right') {
                        rightHandId = hand['id'];
                        palmName = "rightPalmPosition";
                    }
                    else {
                        leftHandId = hand['id'];
                        palmName = "leftPalmPosition";
                    }
                    let palmPosition = hand['palmPosition'];
                    points[palmName] = new Point(palmPosition[0], palmPosition[1],
                    ↪  palmPosition[2], frame['timestamp']);
                }
                for (const pointable of frame.pointables) {
                    if (!pointable.tool) {
                        for (const fingerArticulation of fingerArticulations) {
                            // Get label (e.g., rightIndexTipPosition)
                            let side = pointable.handId == rightHandId ? "right" :
                            ↪  "left";
                            let label = `${side}${fingerNames[pointable.type]}$
                            ↪  {fingerArticulation}Position`;
                            let position = pointable[
                            ↪  `${fingerArticulation.toLowerCase()}Position`];
                            points[label] = new Point(position[0], position[1],
                            ↪  position[2], frame.timestamp);
```

```
                        }
                    }
                }
                // Add points to paths (with default point w/ coordinates (x=0,
                 ↪  y=0, z=0))
                Object.keys(points).forEach((articulation) => {
                    let stroke = gestureData.paths[articulation].strokes[0];
                    if (points[articulation] === null) {
                        // Default coordinates
                        stroke.addPoint(new Point(0, 0, 0, frame['timestamp']));
                    } else {
                        // Coordinates from frame
                        stroke.addPoint(points[articulation]);
                    }
                });
            }
        });
    });
    return gestureSet;
}

module.exports = {
    loadDataset
};
```

uWave Dataset

uwave-loader.js

```
const path = require('path');
const fs = require('fs');

const GestureSet = require('../../../framework/gestures/gesture-set').GestureSet;
const GestureClass =
 ↪  require('../../../framework/gestures/gesture-class').GestureClass;
const StrokeData = require('../../../framework/gestures/stroke-data').StrokeData;
const Stroke = require('../../../framework/gestures/stroke-data').Stroke;
const Path = require('../../../framework/gestures/stroke-data').Path;
const Point = require('../../../framework/gestures/point').Point3D;


function loadDataset(name, directory) {
    let gestureSet = new GestureSet(name);
    let dirPath = path.join(directory, name);

    let gestureClasses = {};

    // Browse main directory
    fs.readdirSync(dirPath).forEach((dir) => {
        let gestureClassDirPath = path.join(dirPath, dir);
```

```
        // Subdirectories
        fs.readdirSync(gestureClassDirPath).forEach((file) => {
            let rawGesturePath = path.join(gestureClassDirPath, file);
            let lines = fs.readFileSync(rawGesturePath, 'utf-8').split(/\r?\n/);
            let gestureData = new StrokeData();
            let strokePath = new Path("main");
            let stroke = new Stroke(0);
            strokePath.addStroke(stroke);

            let gestureId = file.slice(23, 24);

            let index = 0;
            for (const line of lines) {
                let data = line.split(' ');
                if(data.length == 3) {
                    let x = parseFloat(data[0]);
                    let y = parseFloat(data[1]);
                    let z = parseFloat(data[2]);
                    let t = index++;
                    stroke.addPoint(new Point(x, y, z, t));
                }
            }

            if (index < 10) {
                // Ignore, probably bad data
                console.log(`Too few datapoints (${index}) - ${rawGesturePath}`);
            } else {
                gestureData.addPath("main", strokePath);
                if (!gestureClasses.hasOwnProperty(gestureId)) {
                    gestureClasses[gestureId] = new GestureClass(`Gesture
                    ↪  ${gestureId}`);
                }
                gestureClasses[gestureId].addSample(gestureData);
            }
        });
    })

    Object.keys(gestureClasses).forEach((key) => {
        gestureSet.addGestureClass(gestureClasses[key]);
    });

    return gestureSet;
}

module.exports = {
    loadDataset
};
```

F.4 Classifiers

F.4.1 GPSDa

gpsda-static-recognizer.js

```
class Recognizer {
    static name = "GPSDAlphaRecognizer";

    constructor(numPoints, alpha = 1) {
        this.alpha = alpha;
        this.numPoints = numPoints;
        this.templates = [];
    }

    recognize(points) {
        if (points.length != this.numPoints) {
            return { success: false, name: 'No match', time: 0.0 };
        }
        var t0 = Date.now();
        var candidate = new CPS("", points, this.alpha);
        var bestTemplate = null;
        var bestDistance = +Infinity;
        for (const template of this.templates) {
            var distance = computeGPSD(candidate.cps, template.cps);
            if (distance < bestDistance) {
                bestTemplate = template;
                bestDistance = distance;
            }
        }
        var t1 = Date.now();
        return (bestTemplate === null) ? { success: false, name: 'No match', time:
         ↪  t1-t0 } : { success: true, name: bestTemplate.name, time: t1-t0 };
    }

    addGesture(name, points) {
        if (points.length != this.numPoints) {
            return false;
        }
        this.templates.push(new CPS(name, points, this.alpha));
        return true;
    }

    removeGesture(name) {
                this.templates = this.templates.filter(cps => cps.name !== name);
        }
}

/**
 * Complete point shape descriptor
 */
```

```javascript
class CPS {
    constructor(name, points, alpha) {
        this.name = name;
        this.cps = computeCPS(points, alpha);
    }
}

class Point {
    constructor(x, y, z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }
}

/**
 * Compute the Global Point Shape Distance (distance between the 2 complete point shapes)
 */
function computeGPSD(cps1, cps2) {
    var distance = 0.0;
    for (var i = 0; i < cps1.length; i++) {
        distance += Math.abs(cps2[i] - cps1[i]);
    }
    return distance;
}

/**
 * Compute the Complete Point Shape descriptor of a set of points
 */
function computeCPS(points, alpha) {
    let barycentre = getBarycentre(points);
    let avgDistance = getAvgAlphaDistance(barycentre, points, alpha);
    let cps = [];
    for (var i = 0; i < points.length - 1; i++) {
        for (var j = i + 1; j < points.length; j++) {
            let alphaDistance = Math.pow(getDistance(points[i], points[j]), alpha);
            cps.push(alphaDistance / avgDistance);
        }
    }
    cps.sort();
    return cps;
}

/**
 * Compute the barycentre of a set of points
 */
function getBarycentre(points) {
    var x = 0.0, y = 0.0, z = 0.0;
    for (const point of points) {
        x += point.x;
```

```
        y += point.y;
        z += point.z;
    }
    x /= points.length;
    y /= points.length;
    z /= points.length;
    return new Point(x, y, z);
}

/**
 * Compute the average distance between a set of points and a reference point
 */
function getAvgAlphaDistance(refPoint, points, alpha) {
    var d = 0.0;
    for (const point of points) {
        d += Math.pow(getDistance(point, refPoint), alpha);
    }
    return d / points.length;
}

/**
 * Compute the euclidean distance between two points
 */
function getDistance(p1, p2) {
    var dx = p2.x - p1.x;
    var dy = p2.y - p1.y;
    var dz = p2.z - p1.z;
        return Math.sqrt(dx * dx + dy * dy + dz * dz);
}

module.exports = {
    Recognizer,
    Point
};
```

classifier.js

```
const AbstractClassifier =
 ↪  require('../../../framework/classifiers/abstract-classifier').AbstractClassifier;
const GPSDaRecognizer = require('./gpsda/gpsda-static-recognizer').Recognizer;
const Point = require('./gpsda/gpsda-static-recognizer').Point;

class Classifier extends AbstractClassifier {

    static name = "GPSDaClassifier";

    constructor(options, dataset) {
        super(options);
        this.articulations = options.articulations;
        this.alpha = options.alpha;
        this.staticRecognizer = new GPSDaRecognizer(options.articulations.length,
         ↪  options.alpha);
```

```
        // Load gestures from the dataset
        if (dataset !== undefined) {
                        dataset.getGestureClasses().forEach((gesture) => {
                                gesture.getSamples().forEach(sample => {
                    this.addPose(gesture.name, sample);
                                });
            });
        }
    }

    addPose(name, sample) {
        let frame = sample.frame;
        let points = []
        for (const articulation of this.articulations) {
            points.push(frame.getArticulation(articulation).point);
        }
        this.staticRecognizer.addGesture(name, points);
    }

    removePose(name) {
        this.staticRecognizer.removeGesture(name);
    }

    classify(frame) {
        let points = []
        for (const articulation of this.articulations) {
            points.push(frame.getArticulation(articulation).point);
        }
        let { success, name, time } = this.staticRecognizer.recognize(points);
        return success ? { 'name': name, 'time': time } : { 'name': "", 'time':
         ↪  time };
    }

    toString() {
        return `${Classifier.name} [ alpha = ${this.alpha.toFixed(2)} ]`;
    }
}

module.exports = {
    Classifier,
    Point
}
```

F.4.2 P^3+

recognizer.js

```
//
// Point class
//
```

```
class Point {
        constructor(x, y, z, id, angle = 0.0) {
        this.x = x;
        this.y = y;
        this.z = z;
                this.id = id;
                this.angle = angle; // normalized turning angle, $P+
    }
}

//
// PointCloud class: a point-cloud template
//
function PointCloud(name, points) // constructor
{
        this.Name = name;
        this.Points = Resample(points, NumPoints);
        this.Points = Scale(this.Points);
        this.Points = TranslateTo(this.Points, Origin);
        this.Points = ComputeNormalizedTurningAngles(this.Points); // $P+
}

//
// PDollarPlusRecognizer constants
//
let NumPoints = 6;
const Origin = new Point(0, 0, 0, 0);

//
// PDollarPlusRecognizer class
//
class Recognizer {

        static name = "P3DollarPlusRecognizer";

    constructor(numArticulations = 6) {
                NumPoints = numArticulations;
                this.PointClouds = new Array();
        }

        //
        // The $P+ Point-Cloud Recognizer API begins here -- 3 methods:
        ↪  Recognize(), AddGesture(), DeleteUserGestures()
        //
        recognize(points) {
                if(points.length != NumPoints) {
                        return { success: false, name: 'No match', time: 0.0 };
                }
                let t0 = Date.now();
                var candidate = new PointCloud("", points);
                var u = -1;
```

```
                var b = +Infinity;
                for (var i = 0; i < this.PointClouds.length; i++) // for each point-cloud template
                {
                        var d = Math.min(
                                CloudDistance(candidate.Points, this.PointClouds[i].Points, b),
                                CloudDistance(this.PointClouds[i].Points, candidate.Points, b)
                                ); // $P+
                        if (d < b) {
                                b = d; // best (least) distance
                                u = i; // point-cloud index
                        }
                }
                let t1 = Date.now();
                return (u == -1) ? { success: false, name: 'No match', time: t1-t0 } : { success: true, name: this.PointClouds[u].Name, time: t1-t0 };
        }

        addGesture(name, points) {
                this.PointClouds[this.PointClouds.length] = new PointCloud(name, points);
                var num = 0;
                for (var i = 0; i < this.PointClouds.length; i++) {
                        if (this.PointClouds[i].Name == name)
                                num++;
                }
                return num;
        }

        removeGesture(name) {
                this.PointClouds = this.PointClouds.filter(pointCloud => pointCloud.Name !== name);
        }
}

//
// Private helper functions from here on down
//
function CloudDistance(pts1, pts2, minSoFar) // revised for $P+
{
        var matched = new Array(pts1.length); // pts1.length == pts2.length
        for (var k = 0; k < pts1.length; k++)
                matched[k] = false;
        var sum = 0;
        for (var i = 0; i < pts1.length; i++)
        {
                var index = -1;
                var min = +Infinity;
```

```
                for (var j = 0; j < pts1.length; j++)
                {
                        var d = DistanceWithAngle(pts1[i], pts2[j]);
                        if (d < min) {
                                min = d;
                                index = j;
                        }
                }
                matched[index] = true;
                sum += min;
                if (sum >= minSoFar) return sum; // early abandoning
        }
        for (var j = 0; j < matched.length; j++)
        {
                if (!matched[j]) {
                        var min = +Infinity;
                        for (var i = 0; i < pts1.length; i++) {
                                var d = DistanceWithAngle(pts1[i], pts2[j]);
                                if (d < min)
                                        min = d;
                        }
                        sum += min;
                        if (sum >= minSoFar) return sum; // early abandoning
                }
        }
        return sum;
}

function Resample(points, n)
{
        var I = PathLength(points) / (n - 1); // interval length
        var D = 0.0;
        var newpoints = new Array(points[0]);
        for (var i = 1; i < points.length; i++)
        {
                if (points[i].id == points[i-1].id)
                {
                        var d = Distance(points[i-1], points[i]);
                        if ((D + d) >= I)
                        {
                                var qx = points[i-1].x + ((I - D) / d) *
                                 ↪  (points[i].x - points[i-1].x);
                                var qy = points[i-1].y + ((I - D) / d) *
                                 ↪  (points[i].y - points[i-1].y);
                                var qz = points[i-1].z + ((I - D) / d) *
                                 ↪  (points[i].z - points[i-1].z);
                                var q = new Point(qx, qy, qz, points[i].id);
                                newpoints[newpoints.length] = q; // append new
                                 ↪  point 'q'
                                points.splice(i, 0, q); // insert 'q' at position i
                                 ↪  in points s.t. 'q' will be the next i
```

```
                                D = 0.0;
                        }
                        else D += d;
                }
        }
        if (newpoints.length == n - 1) // sometimes we fall a rounding-error short of adding the last point, so add it if so
                newpoints[newpoints.length] = new Point(points[points.length - 1].x, points[points.length - 1].y, points[points.length - 1].z, points[points.length - 1].id);
        return newpoints;
}

function Scale(points)
{
        var minX = +Infinity, maxX = -Infinity, minY = +Infinity, maxY = -Infinity, minZ = +Infinity, maxZ = -Infinity;
        for (var i = 0; i < points.length; i++) {
                minX = Math.min(minX, points[i].x);
                minY = Math.min(minY, points[i].y);
                minZ = Math.min(minZ, points[i].z);
                maxX = Math.max(maxX, points[i].x);
                maxY = Math.max(maxY, points[i].y);
                maxZ = Math.max(maxZ, points[i].z);
        }
        var size = Math.max(maxX - minX, maxY - minY, maxZ - minZ);
        var newpoints = new Array();
        for (var i = 0; i < points.length; i++) {
                var qx = (points[i].x - minX) / size;
                var qy = (points[i].y - minY) / size;
                var qz = (points[i].z - minZ) / size;
                newpoints[newpoints.length] = new Point(qx, qy, qz, points[i].id);
        }
        return newpoints;
}

function TranslateTo(points, pt) // translates points' centroid
{
        var c = Centroid(points);
        var newpoints = new Array();
        for (var i = 0; i < points.length; i++) {
                var qx = points[i].x + pt.x - c.x;
                var qy = points[i].y + pt.y - c.y;
                var qz = points[i].z + pt.z - c.z;
                newpoints[newpoints.length] = new Point(qx, qy, qz, points[i].id);
        }
        return newpoints;
}

function ComputeNormalizedTurningAngles(points) // $P+
{
```

```
191         var newpoints = new Array();
192         newpoints[0] = new Point(points[0].x, points[0].y, points[0].z,
            ↪  points[0].id); // first point
193         for (var i = 1; i < points.length - 1; i++)
194         {
195                 var dx = (points[i+1].x - points[i].x) * (points[i].x -
                    ↪  points[i-1].x);
196                 var dy = (points[i+1].y - points[i].y) * (points[i].y -
                    ↪  points[i-1].y);
197                 var dz = (points[i+1].z - points[i].z) * (points[i].z -
                    ↪  points[i-1].z);
198                 var dn = Distance(points[i+1], points[i]) * Distance(points[i],
                    ↪  points[i-1]);
199                 var cosangle = Math.max(-1.0, Math.min(1.0, (dx + dy + dz) / dn));
                    ↪  // ensure [-1,+1]
200                 var angle = Math.acos(cosangle) / Math.PI; // normalized angle
201                 newpoints[newpoints.length] = new Point(points[i].x, points[i].y,
                    ↪  points[i].z, points[i].id, angle);
202         }
203         newpoints[newpoints.length] = new Point( // last point
204                 points[points.length - 1].x,
205                 points[points.length - 1].y,
206                 points[points.length - 1].z,
207                 points[points.length - 1].id);
208         return newpoints;
209 }
210
211 function Centroid(points)
212 {
213         var x = 0.0, y = 0.0, z = 0.0;
214         for (var i = 0; i < points.length; i++) {
215                 x += points[i].x;
216                 y += points[i].y;
217                 z += points[i].z;
218         }
219         x /= points.length;
220         y /= points.length;
221         z /= points.length;
222         return new Point(x, y, z, 0);
223 }
224
225 function PathLength(points) // length traversed by a point path
226 {
227         var d = 0.0;
228         for (var i = 1; i < points.length; i++) {
229                 if (points[i].id == points[i-1].id)
230                         d += Distance(points[i-1], points[i]);
231         }
232         return d;
233 }
234
```

```
function DistanceWithAngle(p1, p2) // $P+
{
        var dx = p2.x - p1.x;
        var dy = p2.y - p1.y;
        var dz = p2.z - p1.z;
        var da = p2.angle - p1.angle;
        return Math.sqrt(dx * dx + dy * dy + dz * dz + da * da);
}

function Distance(p1, p2) // Euclidean distance between two points
{
        var dx = p2.x - p1.x;
        var dy = p2.y - p1.y;
        var dz = p2.z - p1.z;
        return Math.sqrt(dx * dx + dy * dy + dz * dz);
}

module.exports = {
        Point,
        Recognizer
};
```

classifier.js

```
const AbstractClassifier =
 ↪  require('../../../framework/classifiers/abstract-classifier').AbstractClassifier;
const P3DollarPlusRecognizer = require('./p3dollarplus/recognizer').Recognizer;
const Point = require('./p3dollarplus/recognizer').Point;

class Classifier extends AbstractClassifier {

    static name = "P3DollarPlusClassifier";

    constructor(options, dataset) {
        super(options);
        this.articulations = options.articulations;
        this.staticRecognizer = new
         ↪  P3DollarPlusRecognizer(this.articulations.length);
        // Load gestures from the dataset
        if (dataset !== undefined) {
                        dataset.getGestureClasses().forEach((gesture) => {
                                gesture.getSamples().forEach(sample => {
                    this.addPose(gesture.name, sample);
                                });
            });
        }
    }

    addPose(name, sample) {
        let frame = sample.frame;
        let points = []
```

```
        for (const articulation of this.articulations) {
            points.push(frame.getArticulation(articulation).point);
        }
        this.staticRecognizer.addGesture(name, points);
    }

    removePose(name) {
        this.staticRecognizer.removeGesture(name);
    }

    classify(frame) {
        let points = []
        for (const articulation of this.articulations) {
            let point = frame.getArticulation(articulation).point;
            points.push(new Point(point.x, point.y, point.z, 0));
        }
        try {
            let { success, name, time } = this.staticRecognizer.recognize(points);
            return success ? { 'name': name, 'time': time } : { 'name': "", 'time':
             ↪  time };
        } catch(err) {
            return { 'name': "", 'time': 0.0 };
        }
    }

    toString() {
        return `${Classifier.name}`;
    }
}

module.exports = {
    Classifier,
    Point
}
```

F.5 Analyzers

F.5.1 Basic Analyzer

basic-analyzer.js

```
const Point = require('../../../framework/gestures/point').Point3D;
const AbstractAnalyzer =
 ↪  require('../../../framework/analyzers/abstract-analyzer').AbstractAnalyzer;

class Analyzer extends AbstractAnalyzer {

    constructor(options) {
        super(options);
        this.previousFrame = null;
    }

    analyze(frame) {
        if (this.previousFrame === null) {
            // Initialize first frame
            this.previousFrame = frame;
        }
        // Compute pinch, rotation, and translation wrt. first frame.
        let pinch = computePinch(this.previousFrame, frame);
        let rotation = computeRotation(this.previousFrame, frame);
        let translation = computeTranslation(this.previousFrame, frame);
        let thumbVector = computeThumbVector(frame);
        // Save the current frame for next call
        this.previousFrame = frame;
        return { 'rotation': rotation, 'pinch': pinch, 'translation': translation,
         ↪  'thumbVector': thumbVector };
    }

    reset() {
        this.previousFrame = null;
    }
}

function computePinch(fromFrame, toFrame) {
    let dFrom =
     ↪  getDistance(fromFrame.getArticulation("rightThumbTipPosition").point,
     ↪  fromFrame.getArticulation("rightIndexTipPosition").point);
    let dTo = getDistance(toFrame.getArticulation("rightThumbTipPosition").point,
     ↪  toFrame.getArticulation("rightIndexTipPosition").point);
    return (dTo / dFrom);
}

function computeRotation(fromFrame, toFrame) {
    let vectorFrom =
     ↪  translateTo(fromFrame.getArticulation("rightIndexTipPosition").point,
     ↪  fromFrame.getArticulation("rightThumbTipPosition").point);
```

```
    let vectorTo =
     ↪  translateTo(toFrame.getArticulation("rightIndexTipPosition").point,
     ↪  toFrame.getArticulation("rightThumbTipPosition").point);
    let a = computeAngle(vectorTo, vectorFrom);
    return a
}

function computeTranslation(fromFrame, toFrame) {
    let pFrom = fromFrame.getArticulation("rightPalmPosition").point;
    let pTo = toFrame.getArticulation("rightPalmPosition").point;
    let dx = pTo.x - pFrom.x;
    let dy = pTo.y - pFrom.y;
    let dz = pTo.z - pFrom.z;
    return [dx, dy, dz];
}

function computeThumbVector(frame) {
    let tVector = translateTo(frame.getArticulation("rightThumbTipPosition").point,
     ↪  frame.getArticulation("rightPalmPosition").point);
    return [tVector.x, tVector.y, tVector.z];
}

function computeAngle(p1, p2) {
    return Math.atan2(p1.x * p2.y - p1.y * p2.x, p1.x * p2.x + p1.y * p2.y);
}

function getDistance(p1, p2) {
    var dx = p2.x - p1.x;
    var dy = p2.y - p1.y;
    var dz = p2.z - p1.z;
        return Math.sqrt(dx * dx + dy * dy + dz * dz);
}

function translateTo(p1, p2) {
        return new Point(p1.x - p2.x, p1.y - p2.y, p1.z - p2.z);
}

module.exports = {
    Analyzer
}
```

F.6 Segmenters

F.6.1 Left-Hand Segmenter

lefthand-segmenter.js

```
const AbstractSegmenter =
 ↪  require('../../framework/segmenters/abstract-segmenter').AbstractSegmenter;
const StrokeData = require('../../framework/gestures/stroke-data').StrokeData;
const Stroke = require('../../framework/gestures/stroke-data').Stroke;
const Path = require('../../framework/gestures/stroke-data').Path;
const Point = require('../../framework/gestures/point').Point3D;

class Segmenter extends AbstractSegmenter {
    constructor(options) {
        super(options);
        this.minFrames = options.minSegmentLength;
        this.maxFrames = options.maxSegmentLength;
        this.strokeData = null;
        this.frameCount = 0;
    }

    segment(frame) {
        if (frame.hasLeftHand) {
            if (this.frameCount >= this.maxFrames) {
                // Max number of frames reached
                let oldStrokeData = this.strokeData;
                this.strokeData = null;
                this.frameCount = 0;
                return oldStrokeData;
            }
            if (this.strokeData === null) {
                // Initialize strokeData
                this.strokeData = new StrokeData();
                for (const articulation of frame.articulations) {
                    let path = new Path(articulation.label);
                    this.strokeData.addPath(articulation.label, path);
                    let stroke = new Stroke();
                    path.addStroke(stroke);
                    // Quick hack to remove left hand
                    if (articulation.label.includes("left")) {
                        stroke.addPoint(new Point(0, 0, 0, frame.timestamp));
                    } else {
                        stroke.addPoint(articulation.point);
                    }
                }
            } else {
                for (const articulation of frame.articulations) {
                    let path = this.strokeData.paths[articulation.label];
                    let stroke = path.strokes[0];
                    // Quick hack to remove left hand
```

```
                    if (articulation.label.includes("left")) {
                        stroke.addPoint(new Point(0, 0, 0, frame.timestamp));
                    } else {
                        stroke.addPoint(articulation.point);
                    }
                }
            }
            this.frameCount++;
        } else if (this.frameCount > this.minFrames) {
            // Left hand removed and enough frames
            let oldStrokeData = this.strokeData;
            this.strokeData = null;
            this.frameCount = 0;
            return oldStrokeData;
        } else {
            // Left hand removed and not enough frames
            this.strokeData = null;
            this.frameCount = 0;
        }
        return null;
    }

    notifyRecognition() {
        // Do nothing
    }
}

module.exports = {
    Segmenter
};
```

F.6.2 Zoning Segmenter

zoning-segmenter.js

```
const AbstractSegmenter =
 ↪  require('../../framework/segmenters/abstract-segmenter').AbstractSegmenter;
const StrokeData = require('../../framework/gestures/stroke-data').StrokeData;
const Stroke = require('../../framework/gestures/stroke-data').Stroke;
const Path = require('../../framework/gestures/stroke-data').Path;

const xBound = 120;
const zBound = 60;

class Segmenter extends AbstractSegmenter {
    constructor(options) {
        super(options);
        this.minFrames = options.minSegmentLength;
        this.maxFrames = options.maxSegmentLength;
        this.numberPauseFrames = options.pauseLength;
```

```
        this.strokeData = null;
        this.pauseCount = 0;
    }

    segment(frame) {
        // Increment pause count
        this.pauseCount = Math.max(this.pauseCount - 1, 0);
        if (this.pauseCount != 0) {
            return null;
        }
        if (isWithinBounds(frame)) {
            // At least one hand is in the zone
            if (this.frameCount >= this.maxFrames) {
                // Max number of frames reached
                let oldStrokeData = this.strokeData;
                this.strokeData = null;
                this.frameCount = 0;
                return oldStrokeData;
            }
            if (this.strokeData === null) {
                // Initialize strokeData
                this.strokeData = new StrokeData();
                for (const articulation of frame.articulations) {
                    let path = new Path(articulation.label);
                    this.strokeData.addPath(articulation.label, path);
                    let stroke = new Stroke();
                    path.addStroke(stroke);
                    stroke.addPoint(articulation.point);
                }
            } else {
                for (const articulation of frame.articulations) {
                    let path = this.strokeData.paths[articulation.label];
                    let stroke = path.strokes[0];
                    stroke.addPoint(articulation.point);
                }
            }
            this.frameCount++;
        } else if (this.frameCount > this.minFrames) {
            // Hands outside of the zone & enough frames
            let oldStrokeData = this.strokeData;
            this.strokeData = null;
            this.frameCount = 0;
            return oldStrokeData;
        } else {
            // Hands outside of the zone and not enough frames
            this.strokeData = null;
            this.frameCount = 0;
        }
        return null;
    }

```

```
    notifyRecognition() {
        this.pauseCount = this.numberPauseFrames;
    }
}

function isWithinBounds(frame) {
    let withinBounds = false;
    if (frame.hasLeftHand) {
        let x = frame.getArticulation("leftPalmPosition").point.x;
        let z = frame.getArticulation("leftPalmPosition").point.z;
        withinBounds = x < xBound && x > -xBound && z < zBound && z > -zBound;
    }
    if (!withinBounds && frame.hasRightHand) {
        let x = frame.getArticulation("rightPalmPosition").point.x;
        let z = frame.getArticulation("rightPalmPosition").point.z;
        withinBounds = x < xBound && x > -xBound && z < zBound && z > -zBound;
    }
    return withinBounds;
}

module.exports = {
    Segmenter
};
```

F.6.3 Window Segmenter

window-segmenter.js

```
const AbstractSegmenter =
 ↪  require('../../framework/segmenters/abstract-segmenter').AbstractSegmenter;
const StrokeData = require('../../framework/gestures/stroke-data').StrokeData;
const Stroke = require('../../framework/gestures/stroke-data').Stroke;
const Path = require('../../framework/gestures/stroke-data').Path;

class Segmenter extends AbstractSegmenter {

    constructor(options) {
        super(options);
        this.windowWidth = options.windowWidth;
        this.numberIntervalFrames = options.intervalLength;
        this.numberPauseFrames = options.pauseLength;
        this.motionThreshold = options.motionThreshold;
        this.frameBuffer = null;
        this.intervalCount = 0;
        this.pauseCount = 0;
        this.bufferLength = 0;
    }

    segment(frame) {
        if (this.frameBuffer === null) {
```

```
            // Initialize frameBuffer
            this.frameBuffer = initBuffer(frame);
        } else {
            for (const articulation of frame.articulations) {
                // Add the position of this articulation to the buffer
                this.frameBuffer[articulation.label].push(articulation.point);
                if (this.bufferLength >= this.windowWidth) {
                    // Shift items in buffer
                    this.frameBuffer[articulation.label].shift();
                }
            }
        }
        // Increment pause count
        this.pauseCount = Math.max(this.pauseCount - 1, 0);
        this.intervalCount = (this.intervalCount + 1) % this.numberIntervalFrames;
        if (this.bufferLength < this.windowWidth) {
            // Buffer not full
            this.bufferLength++;
        } else if (this.pauseCount == 0 && this.intervalCount == 0 &&
        ↪  (frame.hasLeftHand || frame.hasRightHand)) {
            // Buffer full & ready
            if (isMotion(this.frameBuffer, this.motionThreshold,
            ↪  ["rightPalmPosition", "rightThumbTipPosition",
            ↪  "rightIndexTipPosition"])) {
                // Create the StrokeData object from the frames in the buffer
                let strokeData = new StrokeData();
                for (const articulation of frame.articulations) {
                    let path = new Path(articulation.label);
                    strokeData.addPath(articulation.label, path);
                    let stroke = new Stroke();
                    path.addStroke(stroke);
                    stroke.points = this.frameBuffer[articulation.label].slice();
                }
                return strokeData;
            }
        }
        return null;
    }

    notifyRecognition() {
        this.pauseCount = this.numberPauseFrames;
    }
}

function initBuffer(frame) {
    let buffer = {};
    for (const articulation of frame.articulations) {
        buffer[articulation.label] = [ articulation.point ];
    }
    return buffer;
}
```

```
function isMotion(frameBuffer, threshold, articulations) {
    for (const articulation of articulations) {
        // Compute motion related to first point
        let refPoint = frameBuffer[articulation][0];
        for (let i = frameBuffer[articulation].length - 1; i > 0; i--) {
            let motion = distance(refPoint, frameBuffer[articulation][i]);
            if (motion >= threshold) {
                return true;
            }
        }
    }
    return false;
}

function distance(p1, p2) {
        var dx = p2.x - p1.x;
        var dy = p2.y - p1.y;
        var dz = p2.z - p1.z;
        return Math.sqrt(dx * dx + dy * dy + dz * dz);
}

module.exports = {
    Segmenter
};
```

F.7 Recognizers

F.7.1 3 cent

recognizer.js

```
const AbstractRecognizer =
 ↪  require('../../../framework/recognizers/abstract-recognizer').AbstractRecognizer;
const math = require('mathjs');
const Spline = require('cubic-spline');


class Point {
        constructor(x, y, z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }
}


let NumPoints = 8;
let pathName;

class Recognizer extends AbstractRecognizer {

    static name = "ThreeCentRecognizer";

    constructor(options, dataset) {
                super();
        NumPoints = options.samplingPoints;
        pathName = options.pathName;
        this.templates = {};
        this.threshold = Infinity;
                if (dataset!==undefined){
                        dataset.getGestureClasses().forEach((gesture) => {
                                gesture.getSamples().forEach(sample => {
                                                this.addGesture(gesture.name,
                                                 ↪  sample);
                                        }
                                );
                        });
                }
    }

    addGesture(name, sample) {
        let points = convert(sample);
        points = normalizeP(points, NumPoints);
        if (!this.templates.hasOwnProperty(name)) {
            this.templates[name] = [];
        }
```

```
        this.templates[name].push(points);
        let distances = new Array();
        // For each class of gesture
        Object.keys(this.templates).forEach((name) => {
            // Compute the distance between each pair of gesture in the class, add
             ↪  it to distances
            for (let i = 0; i < this.templates[name].length; i++) {
                for (let k = i+1; k < this.templates[name].length; k++) {
                    distances.push(gestureDistance(this.templates[name][i],
                     ↪  this.templates[name][k]));
                }
            }
        });

        if (distances.length > 0) {
            // Remove outliers and sort distances
            //distances = rmOutliers(distances);
            distances = distances.sort(function(a, b){return a - b});
            // Histogram heuristic to compute decision distance threshold
            var hist = histogram(distances, 30);
            var max = Math.max(...hist["values"]);
            var indexMax = hist["values"].indexOf(max);
            var binWidth = hist["binWidth"];
            var distancesHist = (((indexMax) * binWidth) + ((indexMax + 1) *
             ↪  binWidth)) / 2;
            this.threshold = distancesHist;
        }
    }

    removeGesture(name) {
        console.log("Cannot remove gesture!");
    }

    recognize(sample) {
                let points = convert(sample);
        let t0 = Date.now();
        points = normalizeP(points, NumPoints);
        let bestFitClass = "";
        //let minDist = this.threshold;
        let minDist = Infinity;
        // Compare gesture w/ each template
        Object.keys(this.templates).forEach((name) => {
            for (let i = 0; i < this.templates[name].length; i++) {
                let tmpDist = gestureDistance(points, this.templates[name][i]);
                if (tmpDist < minDist) {
                    minDist = tmpDist;
                    bestFitClass = name;
                }
            }
        });
        let t1 = Date.now();
```

```
                return (bestFitClass === "") ? { name: "", time: t1-t0, 'Score':
                 ↪  0.0 } : { name: bestFitClass, time: t1-t0, 'Score': minDist };
        }

    toString() {
        return `${Recognizer.name} [ samplingPoints = ${NumPoints}, pathName =
         ↪  ${pathName} ]`;
    }
}

function convert(sample) {
    return sample.paths[pathName].strokes[0].points;
}

function histogram(data, numBins) {
    let binWidth = math.round((math.max(data) / numBins) + 0.0005, 3);
    let currentItemId = 0;
    var bins = new Array(numBins).fill(0);
    for (let i = 0; i < numBins; i++) {
        while (currentItemId < math.size(data)[0] && (data[currentItemId] < (i + 1)
         ↪  * binWidth || i == numBins - 1)) {
            bins[i] += 1;
            currentItemId += 1;
        }
    }
    return { "values": bins, "binWidth": binWidth };
}

function gestureDistance(points1, points2) {
    let dq = 0;
    for (var i = 0; i < points1.length; i++) {
        var dx = points1[i].x - points2[i].x;
        var dy = points1[i].y - points2[i].y;
        var dz = points1[i].z - points2[i].z;
        dq += dx * dx + dy * dy + dz * dz;
    }
    return dq;
}

function normalizeP(points, n) {
    let dist = [0];
    for (let i = 1; i < points.length; i++) {
        dx = points[i].x - points[i-1].x;
        dy = points[i].y - points[i-1].y;
        dz = points[i].z - points[i-1].z;
        dist.push(Math.sqrt(dx * dx + dy * dy + dz * dz));
    }
    var s = cumsum(dist);
    var len = s[s.length - 1];
    // Resample
    let h = (points.length-1) / (n - 1);
```

```
    let xq = math.range(0, points.length - 1, h)._data;
    xq.push(points.length - 1);
    let x = [];
    let xAxis = [];
    let yAxis = [];
    let zAxis = [];
    for (let i = 0; i < points.length; i++) {
        x.push(i);
        xAxis.push(points[i].x);
        yAxis.push(points[i].y);
        zAxis.push(points[i].z);
    }
    let sX = spline(x, xAxis, xq);
    let sY = spline(x, yAxis, xq);
    let sZ = spline(x, zAxis, xq);
    let avgX = sX.reduce((a,b) => a + b, 0) / sX.length;
    let avgY = sY.reduce((a,b) => a + b, 0) / sY.length;
    let avgZ = sZ.reduce((a,b) => a + b, 0) / sZ.length;
    let newPoints = [];
    for (let i = 0; i < xq.length; i++) {
        // Center (wrt. centroid) and scale
        newPoints.push(new Point((sX[i] - avgX) / len, (sY[i] - avgY) / len, (sZ[i]
        ↪ - avgZ) / len));
    }
    return newPoints;
}

function cumsum(data) {
    var acc = 0;
    return math.map(data, function(x) {
        acc += x;
        return acc;
    })
}

// cubic interpolation of x and y.
// xq: query points
function spline(x, y, xq) {
    const spline = new Spline(x, y);
    var s = new Array(xq.length);
    for (let i = 0; i < xq.length; i++) {
        s[i] = spline.at(xq[i]);
    }
    return s;
}

module.exports = {
        Recognizer
};
```

F.7.2 P^3

p3dollar.js

```
//
// Point class
//
function Point(x, y, z, id) // constructor
{
        this.X = x;
        this.Y = y;
        this.Z = z;
        this.ID = id; // stroke ID to which this point belongs (1,2,3,etc.)
}
//
// PointCloud class: a point-cloud template
//
function PointCloud(name, points) // constructor
{
        this.Name = name;
        this.Points = Resample(points, NumPoints);
        this.Points = Scale(this.Points);
        this.Points = TranslateTo(this.Points, Origin);
}
//
// Result class
//
function Result(name, score, ms) // constructor
{
        this.Name = name;
        this.Score = score;
        this.Time = ms;
}
//
// P3DollarRecognizer constants
//
let NumPoints = 32;
const Origin = new Point(0, 0, 0, 0);
//
// P3DollarRecognizer class
//
function P3DollarRecognizer(numPoints) // constructor
{
        this.PointClouds = new Array();
        NumPoints = numPoints;
        //
        // The $P3 Point-Cloud Recognizer API begins here -- 4 methods:
        ↪  Recognize(), AddGesture(), RemoveGesture(), DeleteUserGestures()
        //
        this.Recognize = function(points)
        {
                var t0 = Date.now();
```

```
48                  var candidate = new PointCloud("", points);
49
50                  var u = -1;
51                  var b = +Infinity;
52                  for (var i = 0; i < this.PointClouds.length; i++) // for each
                     ↪  point-cloud template
53                  {
54                          var d = GreedyCloudMatch(candidate.Points,
                             ↪  this.PointClouds[i]);
55                          if (d < b) {
56                                  b = d; // best (least) distance
57                                  u = i; // point-cloud index
58                          }
59                  }
60                  var t1 = Date.now();
61                  return (u == -1) ? new Result("No match.", 0.0, t1-t0) : new
                     ↪  Result(this.PointClouds[u].Name, b > 1.0 ? 1.0 / b : 1.0,
                     ↪  t1-t0);
62          }
63          this.AddGesture = function(name, points)
64          {
65                  this.PointClouds[this.PointClouds.length] = new PointCloud(name,
                     ↪  points);
66                  var num = 0;
67                  for (var i = 0; i < this.PointClouds.length; i++) {
68                          if (this.PointClouds[i].Name == name)
69                                  num++;
70                  }
71                  return num;
72      }
73      this.RemoveGesture = function(name)
74      {
75                  this.PointClouds = this.PointClouds.filter(pointCloud =>
                     ↪  pointCloud.Name !== name);
76          }
77          this.DeleteUserGestures = function()
78          {
79                  this.PointClouds.length = NumPointClouds; // clears any beyond the
                     ↪  original set
80                  return NumPointClouds;
81          }
82  }
83  //
84  // Private helper functions from here on down
85  //
86  function GreedyCloudMatch(points, P)
87  {
88          var e = 0.50;
89          var step = Math.floor(Math.pow(points.length, 1.0 - e));
90          var min = +Infinity;
91          for (var i = 0; i < points.length; i += step) {
```

```
                var d1 = CloudDistance(points, P.Points, i);
                var d2 = CloudDistance(P.Points, points, i);
                min = Math.min(min, Math.min(d1, d2)); // min3
        }
        return min;
}
function CloudDistance(pts1, pts2, start)
{
        var matched = new Array(pts1.length); // pts1.length == pts2.length
        for (var k = 0; k < pts1.length; k++)
                matched[k] = false;
        var sum = 0;
        var i = start;
        do
        {
                var index = -1;
                var min = +Infinity;
                for (var j = 0; j < matched.length; j++)
                {
                        if (!matched[j]) {
                                var d = Distance(pts1[i], pts2[j]);
                                if (d < min) {
                                        min = d;
                                        index = j;
                                }
                        }
                }
                matched[index] = true;
                var weight = 1 - ((i - start + pts1.length) % pts1.length) /
                 ↪  pts1.length;
                sum += weight * min;
                i = (i + 1) % pts1.length;
        } while (i != start);
        return sum;
}
function Resample(points, n)
{
        var I = PathLength(points) / (n - 1); // interval length
        var D = 0.0;
        var newpoints = new Array(points[0]);
        for (var i = 1; i < points.length; i++)
        {
                if (points[i].ID == points[i-1].ID)
                {
                        var d = Distance(points[i-1], points[i]);
                        if ((D + d) >= I)
                        {
                                var qx = points[i-1].X + ((I - D) / d) *
                                 ↪  (points[i].X - points[i-1].X);
                                var qy = points[i-1].Y + ((I - D) / d) *
                                 ↪  (points[i].Y - points[i-1].Y);
```

```
                                var qz = points[i-1].Z + ((I - D) / d) *
                                 ↪  (points[i].Z - points[i-1].Z);
                                var q = new Point(qx, qy, qz, points[i].ID);
                                newpoints[newpoints.length] = q; // append new
                                 ↪  point 'q'
                                points.splice(i, 0, q); // insert 'q' at position i
                                 ↪  in points s.t. 'q' will be the next i
                                D = 0.0;
                        }
                        else D += d;
                }
        }
        if (newpoints.length == n - 1) // sometimes we fall a rounding-error short
         ↪  of adding the last point, so add it if so
                newpoints[newpoints.length] = new Point(points[points.length -
                 ↪  1].X, points[points.length - 1].Y, points[points.length - 1].Z,
                 ↪  points[points.length - 1].ID);
        return newpoints;
}
function Scale(points)
{
        var minX = +Infinity, maxX = -Infinity, minY = +Infinity, maxY = -Infinity,
         ↪  minZ = +Infinity, maxZ = -Infinity;
        for (var i = 0; i < points.length; i++) {
                minX = Math.min(minX, points[i].X);
        minY = Math.min(minY, points[i].Y);
        minZ = Math.min(minZ, points[i].Z);
                maxX = Math.max(maxX, points[i].X);
        maxY = Math.max(maxY, points[i].Y);
        maxZ = Math.max(maxZ, points[i].Z);
        }
        var size = Math.max(maxX - minX, maxY - minY, maxZ - minZ);
        var newpoints = new Array();
        for (var i = 0; i < points.length; i++) {
                var qx = (points[i].X - minX) / size;
                var qy = (points[i].Y - minY) / size;
                var qz = (points[i].Z - minZ) / size;
                newpoints[newpoints.length] = new Point(qx, qy, qz, points[i].ID);
        }
        return newpoints;
}
function TranslateTo(points, pt) // translates points' centroid to pt
{
        var c = Centroid(points);
        var newpoints = new Array();
        for (var i = 0; i < points.length; i++) {
                var qx = points[i].X + pt.X - c.X;
                var qy = points[i].Y + pt.Y - c.Y;
                newpoints[newpoints.length] = new Point(qx, qy, points[i].ID);
        }
        return newpoints;
```

```
}
function Centroid(points)
{
        var x = 0.0, y = 0.0, z = 0.0;
        for (var i = 0; i < points.length; i++) {
                x += points[i].X;
        y += points[i].Y;
        z += points[i].Z;
        }
        x /= points.length;
    y /= points.length;
    z /= points.length;
        return new Point(x, y, z, 0);
}
function PathLength(points) // length traversed by a point path
{
        var d = 0.0;
        for (var i = 1; i < points.length; i++) {
                if (points[i].ID == points[i-1].ID)
                        d += Distance(points[i-1], points[i]);
        }
        return d;
}
function Distance(p1, p2) // Euclidean distance between two points
{
        var dx = p2.X - p1.X;
        var dy = p2.Y - p1.Y;
        var dz = p2.Z - p1.Z;
        return Math.sqrt(dx * dx + dy * dy + dz * dz);
}

module.exports = {
    P3DollarRecognizer,
    Point
}
```

recognizer.js

```
const AbstractRecognizer =
 ↪  require('../../../framework/recognizers/abstract-recognizer').AbstractRecognizer;
const P3DollarRecognizer = require('./p3dollar/p3dollar').P3DollarRecognizer;
const Point = require('./p3dollar/p3dollar').Point;

class Recognizer extends AbstractRecognizer {

        static name = "P3DollarRecognizer";

    constructor(options, dataset) {
                super();
                this.samplingPoints = options.samplingPoints;
        this.articulations = options.articulations;
```

```
13          this.recognizer = new P3DollarRecognizer(this.samplingPoints);
14                  if (dataset !== undefined){
15                          dataset.getGestureClasses().forEach((gesture) => {
16                                  gesture.getSamples().forEach(sample => {
17                                                  this.addGesture(gesture.name,
                                                    ↪  sample);
18                                          }
19                                  );
20                          });
21                  }
22          }
23
24          addGesture(name, sample) {
25                  let points = convert(sample, this.articulations);
26                  this.recognizer.AddGesture(name, points);
27          }
28
29      removeGesture(name) {
30                  this.recognizer.RemoveGesture(name);
31          }
32
33      recognize(sample) {
34                  let points = convert(sample, this.articulations);
35                  if(points.length === 0) {
36              return { name: "", time: 0, score: 0.0 };
37          }
38          let result = this.recognizer.Recognize(points);
39                  return (result.Name === "No match.") ? { name: "", time:
                    ↪  result.Time, score: result.Score } : { name: result.Name, time:
                    ↪  result.Time, score: result.Score };
40          }
41
42          toString() {
43          return `${Recognizer.name} [ samplingPoints = ${this.samplingPoints},
             ↪  articulations = ${this.articulations} ]`;
44      }
45
46  }
47
48  function convert(sample, articulations) {
49          let points = [];
50          articulations.forEach((articulation, articulationID) => {
51                  sample.paths[articulation].strokes.forEach((stroke, strokeId) => {
52                          stroke.points.forEach((point) => {
53                                  // If multipoint, one stroke per articulation,
                                    ↪  otherwise, keep original strokes
54                                  let index = articulations.length > 1 ?
                                    ↪  articulationID : strokeId;
55                                  points.push(new Point(point.x, point.y, point.z,
                                    ↪  index));
56                          });
```

```
                });
        });
        return points;
}


module.exports = {
        Recognizer
};
```

F.7.3 P^3+

p3dollarplus.js

```
//
// Point class
//
function Point(x, y, z, id, angle = 0.0)
{
        this.X = x;
    this.Y = y;
    this.Z = z;
        this.ID = id;
        this.Angle = angle; // normalized turning angle, $P+
}
//
// PointCloud class: a point-cloud template
//
function PointCloud(name, points) // constructor
{
        this.Name = name;
        this.Points = Resample(points, NumPoints);
        this.Points = Scale(this.Points);
        this.Points = TranslateTo(this.Points, Origin);
        this.Points = ComputeNormalizedTurningAngles(this.Points); // $P+
}
//
// Result class
//
function Result(name, score, ms) // constructor
{
        this.Name = name;
        this.Score = score;
        this.Time = ms;
}
//
// PDollarPlusRecognizer constants
//
var NumPoints = 32;
const Origin = new Point(0,0,0,0);
```

```
37  //
38  // PDollarPlusRecognizer class
39  //
40  function P3DollarPlusRecognizer(numPoints) // constructor
41  {
42      this.PointClouds = new Array();
43      NumPoints = numPoints;
44          //
45          // The $P+ Point-Cloud Recognizer API begins here -- 4 methods:
            ↪  Recognize(), AddGesture(), RemoveGesture(), DeleteUserGestures()
46          //
47          this.Recognize = function(points)
48          {
49                  var t0 = Date.now();
50                  var candidate = new PointCloud("", points);
51
52                  var u = -1;
53                  var b = +Infinity;
54                  for (var i = 0; i < this.PointClouds.length; i++) // for each
                    ↪  point-cloud template
55                  {
56                          var d = Math.min(
57                                  CloudDistance(candidate.Points,
                                    ↪  this.PointClouds[i].Points, b),
58                                  CloudDistance(this.PointClouds[i].Points,
                                    ↪  candidate.Points, b)
59                                  ); // $P+
60                          if (d < b) {
61                                  b = d; // best (least) distance
62                                  u = i; // point-cloud index
63                          }
64                  }
65                  var t1 = Date.now();
66                  return (u == -1) ? new Result("No match.", 0.0, t1-t0) : new
                    ↪  Result(this.PointClouds[u].Name, b > 1.0 ? 1.0 / b : 1.0,
                    ↪  t1-t0);
67          }
68          this.AddGesture = function(name, points)
69          {
70                  this.PointClouds[this.PointClouds.length] = new PointCloud(name,
                    ↪  points);
71                  var num = 0;
72                  for (var i = 0; i < this.PointClouds.length; i++) {
73                          if (this.PointClouds[i].Name == name)
74                                  num++;
75                  }
76                  return num;
77      }
78      this.RemoveGesture = function(name)
79      {
80                  this.PointClouds = this.PointClouds.filter(pointCloud =>
                    ↪  pointCloud.Name !== name);
```

```
81          }
82          this.DeleteUserGestures = function()
83          {
84                  this.PointClouds.length = NumPointClouds; // clears any beyond the
                    ↪  original set
85                  return NumPointClouds;
86          }
87  }
88  //
89  // Private helper functions from here on down
90  //
91  function CloudDistance(pts1, pts2, minSoFar) // revised for $P+
92  {
93          var matched = new Array(pts1.length); // pts1.length == pts2.length
94          for (var k = 0; k < pts1.length; k++)
95                  matched[k] = false;
96          var sum = 0;
97          for (var i = 0; i < pts1.length; i++)
98          {
99                  var index = -1;
100                 var min = +Infinity;
101                 for (var j = 0; j < pts1.length; j++)
102                 {
103                         var d = DistanceWithAngle(pts1[i], pts2[j]);
104                         if (d < min) {
105                                 min = d;
106                                 index = j;
107                         }
108                 }
109                 matched[index] = true;
110         sum += min;
111         if (sum >= minSoFar) return sum; // early abandoning
112         }
113         for (var j = 0; j < matched.length; j++)
114         {
115                 if (!matched[j]) {
116                         var min = +Infinity;
117                         for (var i = 0; i < pts1.length; i++) {
118                                 var d = DistanceWithAngle(pts1[i], pts2[j]);
119                                 if (d < min)
120                                         min = d;
121                         }
122             sum += min;
123             if (sum >= minSoFar) return sum; // early abandoning
124                 }
125         }
126         return sum;
127 }
128 function Resample(points, n)
129 {
130         var I = PathLength(points) / (n - 1); // interval length
```

```
131         var D = 0.0;
132         var newpoints = new Array(points[0]);
133         for (var i = 1; i < points.length; i++)
134         {
135                 if (points[i].ID == points[i-1].ID)
136                 {
137                         var d = Distance(points[i-1], points[i]);
138                         if ((D + d) >= I)
139                         {
140                                 var qx = points[i-1].X + ((I - D) / d) *
                                    ↪ (points[i].X - points[i-1].X);
141                                 var qy = points[i-1].Y + ((I - D) / d) *
                                    ↪ (points[i].Y - points[i-1].Y);
142                 var qz = points[i-1].Z + ((I - D) / d) * (points[i].Z -
                    ↪ points[i-1].Z);
143                 var q = new Point(qx, qy, qz, points[i].ID);
144                                 newpoints[newpoints.length] = q; // append new
                                    ↪ point 'q'
145                                 points.splice(i, 0, q); // insert 'q' at position i
                                    ↪ in points s.t. 'q' will be the next i
146                                 D = 0.0;
147                         }
148                         else D += d;
149                 }
150         }
151         if (newpoints.length == n - 1) // sometimes we fall a rounding-error short
            ↪ of adding the last point, so add it if so
152                 newpoints[newpoints.length] = new Point(points[points.length -
                    ↪ 1].X, points[points.length - 1].Y, points[points.length - 1].Z,
                    ↪ points[points.length - 1].ID);
153         return newpoints;
154 }
155 function Scale(points)
156 {
157         var minX = +Infinity, maxX = -Infinity, minY = +Infinity, maxY = -Infinity,
            ↪ minZ = +Infinity, maxZ = -Infinity;
158         for (var i = 0; i < points.length; i++) {
159                 minX = Math.min(minX, points[i].X);
160         minY = Math.min(minY, points[i].Y);
161         minZ = Math.min(minZ, points[i].Z);
162                 maxX = Math.max(maxX, points[i].X);
163         maxY = Math.max(maxY, points[i].Y);
164         maxZ = Math.max(maxZ, points[i].Z);
165         }
166         var size = Math.max(maxX - minX, maxY - minY, maxZ - minZ);
167         var newpoints = new Array();
168         for (var i = 0; i < points.length; i++) {
169                 var qx = (points[i].X - minX) / size;
170         var qy = (points[i].Y - minY) / size;
171         var qz = (points[i].Z - minZ) / size;
172                 newpoints[newpoints.length] = new Point(qx, qy, qz, points[i].ID);
```

```
        }
        return newpoints;
}
function TranslateTo(points, pt) // translates points' centroid
{
        var c = Centroid(points);
        var newpoints = new Array();
        for (var i = 0; i < points.length; i++) {
                var qx = points[i].X + pt.X - c.X;
        var qy = points[i].Y + pt.Y - c.Y;
        var qz = points[i].Z + pt.Z - c.Z;
                newpoints[newpoints.length] = new Point(qx, qy, qz, points[i].ID);
        }
        return newpoints;
}
function ComputeNormalizedTurningAngles(points) // $P+
{
        var newpoints = new Array();
        newpoints[0] = new Point(points[0].X, points[0].Y, points[0].Z,
        ↪  points[0].ID); // first point
        for (var i = 1; i < points.length - 1; i++)
        {
                var dx = (points[i+1].X - points[i].X) * (points[i].X -
                ↪  points[i-1].X);
        var dy = (points[i+1].Y - points[i].Y) * (points[i].Y - points[i-1].Y);
        var dz = (points[i+1].Z - points[i].Z) * (points[i].Z - points[i-1].Z);
                var dn = Distance(points[i+1], points[i]) * Distance(points[i],
                ↪  points[i-1]);
                var cosangle = Math.max(-1.0, Math.min(1.0, (dx + dy + dz) / dn));
                ↪  // ensure [-1,+1]
                var angle = Math.acos(cosangle) / Math.PI; // normalized angle
                newpoints[newpoints.length] = new Point(points[i].X, points[i].Y,
                ↪  points[i].Z, points[i].ID, angle);
        }
        newpoints[newpoints.length] = new Point( // last point
                points[points.length - 1].X,
        points[points.length - 1].Y,
        points[points.length - 1].Z,
                points[points.length - 1].ID);
        return newpoints;
}
function Centroid(points)
{
        var x = 0.0, y = 0.0, z = 0.0;
        for (var i = 0; i < points.length; i++) {
                x += points[i].X;
        y += points[i].Y;
        z += points[i].Z;
        }
        x /= points.length;
    y /= points.length;
```

```
    z /= points.length;
        return new Point(x, y, z, 0);
}
function PathLength(points) // length traversed by a point path
{
        var d = 0.0;
        for (var i = 1; i < points.length; i++) {
                if (points[i].ID == points[i-1].ID)
                        d += Distance(points[i-1], points[i]);
        }
        return d;
}
function DistanceWithAngle(p1, p2) // $P+
{
        var dx = p2.X - p1.X;
    var dy = p2.Y - p1.Y;
    var dz = p2.Z - p1.Z;
        var da = p2.Angle - p1.Angle;
        return Math.sqrt(dx * dx + dy * dy + dz * dz + da * da);
}
function Distance(p1, p2) // Euclidean distance between two points
{
        var dx = p2.X - p1.X;
        var dy = p2.Y - p1.Y;
        var dz = p2.Z - p1.Z;
        return Math.sqrt(dx * dx + dy * dy + dz * dz);
}

module.exports = {
        Point,
        P3DollarPlusRecognizer
};
```

recognizer.js

```
const AbstractRecognizer =
 ↪  require('../../../framework/recognizers/abstract-recognizer').AbstractRecognizer;
const P3DollarPlusRecognizer =
 ↪  require('./p3dollarplus/p3dollarplus').P3DollarPlusRecognizer;
const Point = require('./p3dollarplus/p3dollarplus').Point;

class Recognizer extends AbstractRecognizer {

        static name = "P3DollarPlusRecognizer";

    constructor(options, dataset) {
                super();
                this.samplingPoints = options.samplingPoints;
        this.articulations = options.articulations;
        this.recognizer = new P3DollarPlusRecognizer(this.samplingPoints);
                if (dataset !== undefined){
```

```
                        dataset.getGestureClasses().forEach((gesture) => {
                                gesture.getSamples().forEach(sample => {
                                        this.addGesture(gesture.name,
                                         ↪  sample);
                                }
                                );
                        });
                }
        }

        addGesture(name, sample) {
                let points = convert(sample, this.articulations);
                this.recognizer.AddGesture(name, points);
        }

    removeGesture(name) {
                this.recognizer.RemoveGesture(name);
        }

    recognize(sample) {
                let points = convert(sample, this.articulations);
                if(points.length === 0) {
            return { name: "", time: 0, score: 0.0 };
        }
        let result = this.recognizer.Recognize(points);
                return (result.Name === "No match.") ? { name: "", time:
                 ↪  result.Time, score: result.Score } : { name: result.Name, time:
                 ↪  result.Time, score: result.Score };
        }

        toString() {
        return `${Recognizer.name} [ samplingPoints = ${this.samplingPoints},
         ↪  articulations = ${this.articulations} ]`;
    }

}

function convert(sample, articulations) {
        let points = [];
        articulations.forEach((articulation, articulationID) => {
                sample.paths[articulation].strokes.forEach((stroke, strokeId) => {
                        stroke.points.forEach((point) => {
                                // If multipoint, one stroke per articulation,
                                 ↪  otherwise, keep original strokes
                                let index = articulations.length > 1 ?
                                 ↪  articulationID : strokeId;
                                points.push(new Point(point.x, point.y, point.z,
                                 ↪  index));
                        });
                });
        });
```

```
        return points;
}


module.exports = {
        Recognizer
};
```

F.7.4 $P^3+X

p3dollarplusx.js

```
//
// Point class
//
function Point(x, y, z, id, angle = 0.0)
{
        this.X = x;
    this.Y = y;
    this.Z = z;
        this.ID = id;
        this.Angle = angle; // normalized turning angle, $P+
}
//
// PointCloud class: a point-cloud template
//
function PointCloud(name, points) // constructor
{
        this.Name = name;
        this.Points = Resample(points, NumPoints);
        this.Points = Scale(this.Points);
        this.Points = TranslateTo(this.Points, Origin);
        this.Points = ComputeNormalizedTurningAngles(this.Points); // $P+
}
//
// Result class
//
function Result(name, score, ms) // constructor
{
        this.Name = name;
        this.Score = score;
        this.Time = ms;
}
//
// P3DollarPlusXRecognizer constants
//
var NumPoints = 32;
const Origin = new Point(0,0,0,0);
//
// P3DollarPlusXRecognizer class
```

```
39  //
40  function P3DollarPlusXRecognizer(numPoints) // constructor
41  {
42          this.PointClouds = new Array();
43          this.Conflicts = {};
44          NumPoints = numPoints;
45          //
46          // The $P+ Point-Cloud Recognizer API begins here -- 4 methods:
            ↪  Recognize(), AddGesture(), RemoveGesture(), DeleteUserGestures()
47          //
48          this.Recognize = function(points)
49          {
50                  var t0 = Date.now();
51                  var candidate = new PointCloud("", points);
52                  var {u, b} = this.RecognizeHelper(candidate);
53                  if (u != -1 &&
                     ↪  this.Conflicts.hasOwnProperty(this.PointClouds[u].Name)) {
54                          let dirDist1 = DirDist(candidate.Points,
                             ↪  this.PointClouds[u].Points);
55                          let dirDist2 = DirDist(candidate.Points,
                             ↪  this.PointClouds[this.Conflicts[this.PointClouds[u].Name].Index].
56                          u = dirDist1 > dirDist2 ? u :
                             ↪  this.Conflicts[this.PointClouds[u].Name].Index;
57                  }
58                  var t1 = Date.now();
59                  return (u == -1) ? new Result("No match.", 0.0, t1-t0) : new
                     ↪  Result(this.PointClouds[u].Name, b > 1.0 ? 1.0 / b : 1.0,
                     ↪  t1-t0);
60          }
61          this.AddGesture = function(name, points)
62          {
63                  var template = new PointCloud(name, points);
64                  if (this.PointClouds.length > 0) {
65                          const {u, b} = this.RecognizeHelper(template);
66                          if (u != -1 && this.PointClouds[u].Name != name && (1.0 /
                             ↪  b) > 0.8) {
67                                  this.Conflicts[name] = {'Name':
                                     ↪  this.PointClouds[u].Name, 'Index': u};
68                                  this.Conflicts[this.PointClouds[u].Name] =
                                     ↪  {'Name': name, 'Index':
                                     ↪  this.PointClouds.length};
69                          }
70                  }
71                  this.PointClouds[this.PointClouds.length] = template;
72                  var num = 0;
73                  for (var i = 0; i < this.PointClouds.length; i++) {
74                          if (this.PointClouds[i].Name == name)
75                                  num++;
76                  }
77                  return num;
78      }
```

```
79      this.RemoveGesture = function(name)
80      {
81              this.PointClouds = this.PointClouds.filter(pointCloud =>
                ↪  pointCloud.Name !== name);
82              // Remove entry in Conflicts
83              if (this.Conflicts.hasOwnProperty(name)) {
84                      let otherGestureName = this.Conflicts[name].Name;
85                      delete this.Conflicts[name];
86                      delete this.Conflicts[otherGestureName];
87              }
88      }
89      this.DeleteUserGestures = function()
90      {
91              this.PointClouds.length = NumPointClouds; // clears any beyond the
                ↪  original set
92              return NumPointClouds;
93      }
94      this.RecognizeHelper = function(pointCloud)
95      {
96              var u = -1;
97              var b = +Infinity;
98              for (var i = 0; i < this.PointClouds.length; i++) // for each
                ↪  point-cloud template
99              {
100                     var d = Math.min(
101                             CloudDistance(pointCloud.Points,
                                ↪  this.PointClouds[i].Points, b),
102                             CloudDistance(this.PointClouds[i].Points,
                                ↪  pointCloud.Points, b)
103                             ); // $P+
104                     if (d < b) {
105                             b = d; // best (least) distance
106                             u = i; // point-cloud index
107                     }
108             }
109             return {u, b};
110     }
111 }
112 //
113 // Private helper functions from here on down
114 //
115 function CloudDistance(pts1, pts2, minSoFar) // revised for $P+
116 {
117     var matched = new Array(pts1.length); // pts1.length == pts2.length
118     for (var k = 0; k < pts1.length; k++)
119             matched[k] = false;
120     var sum = 0;
121     for (var i = 0; i < pts1.length; i++)
122     {
123             var index = -1;
124             var min = +Infinity;
```

```
                for (var j = 0; j < pts1.length; j++)
                {
                        var d = DistanceWithAngle(pts1[i], pts2[j]);
                        if (d < min) {
                                min = d;
                                index = j;
                        }
                }
                matched[index] = true;
        sum += min;
        if (sum >= minSoFar) return sum; // early abandoning
        }
        for (var j = 0; j < matched.length; j++)
        {
                if (!matched[j]) {
                        var min = +Infinity;
                        for (var i = 0; i < pts1.length; i++) {
                                var d = DistanceWithAngle(pts1[i], pts2[j]);
                                if (d < min)
                                        min = d;
                        }
            sum += min;
            if (sum >= minSoFar) return sum; // early abandoning
                }
        }
        return sum;
}
function Resample(points, n)
{
        var I = PathLength(points) / (n - 1); // interval length
        var D = 0.0;
        var newpoints = new Array(points[0]);
        for (var i = 1; i < points.length; i++)
        {
                if (points[i].ID == points[i-1].ID)
                {
                        var d = Distance(points[i-1], points[i]);
                        if ((D + d) >= I)
                        {
                                var qx = points[i-1].X + ((I - D) / d) *
                                 ↪  (points[i].X - points[i-1].X);
                                var qy = points[i-1].Y + ((I - D) / d) *
                                 ↪  (points[i].Y - points[i-1].Y);
                var qz = points[i-1].Z + ((I - D) / d) * (points[i].Z -
                 ↪  points[i-1].Z);
                var q = new Point(qx, qy, qz, points[i].ID);
                                newpoints[newpoints.length] = q; // append new
                                 ↪  point 'q'
                                points.splice(i, 0, q); // insert 'q' at position i
                                 ↪  in points s.t. 'q' will be the next i
                                D = 0.0;
```

```
171                         }
172                         else D += d;
173                 }
174         }
175         if (newpoints.length == n - 1) // sometimes we fall a rounding-error short
            ↪  of adding the last point, so add it if so
176                 newpoints[newpoints.length] = new Point(points[points.length -
                    ↪  1].X, points[points.length - 1].Y, points[points.length - 1].Z,
                    ↪  points[points.length - 1].ID);
177         return newpoints;
178 }
179 function Scale(points)
180 {
181         var minX = +Infinity, maxX = -Infinity, minY = +Infinity, maxY = -Infinity,
            ↪  minZ = +Infinity, maxZ = -Infinity;
182         for (var i = 0; i < points.length; i++) {
183                 minX = Math.min(minX, points[i].X);
184         minY = Math.min(minY, points[i].Y);
185         minZ = Math.min(minZ, points[i].Z);
186                 maxX = Math.max(maxX, points[i].X);
187         maxY = Math.max(maxY, points[i].Y);
188         maxZ = Math.max(maxZ, points[i].Z);
189         }
190         var size = Math.max(maxX - minX, maxY - minY, maxZ - minZ);
191         var newpoints = new Array();
192         for (var i = 0; i < points.length; i++) {
193                 var qx = (points[i].X - minX) / size;
194         var qy = (points[i].Y - minY) / size;
195         var qz = (points[i].Z - minZ) / size;
196                 newpoints[newpoints.length] = new Point(qx, qy, qz, points[i].ID);
197         }
198         return newpoints;
199 }
200 function TranslateTo(points, pt) // translates points' centroid
201 {
202         var c = Centroid(points);
203         var newpoints = new Array();
204         for (var i = 0; i < points.length; i++) {
205                 var qx = points[i].X + pt.X - c.X;
206         var qy = points[i].Y + pt.Y - c.Y;
207         var qz = points[i].Z + pt.Z - c.Z;
208                 newpoints[newpoints.length] = new Point(qx, qy, qz, points[i].ID);
209         }
210         return newpoints;
211 }
212 function ComputeNormalizedTurningAngles(points) // $P+
213 {
214         var newpoints = new Array();
215         newpoints[0] = new Point(points[0].X, points[0].Y, points[0].Z,
            ↪  points[0].ID); // first point
216         for (var i = 1; i < points.length - 1; i++)
```

```
        {
                var dx = (points[i+1].X - points[i].X) * (points[i].X -
                 ↪  points[i-1].X);
        var dy = (points[i+1].Y - points[i].Y) * (points[i].Y - points[i-1].Y);
        var dz = (points[i+1].Z - points[i].Z) * (points[i].Z - points[i-1].Z);
                var dn = Distance(points[i+1], points[i]) * Distance(points[i],
                 ↪  points[i-1]);
                var cosangle = Math.max(-1.0, Math.min(1.0, (dx + dy + dz) / dn));
                 ↪  // ensure [-1,+1]
                var angle = Math.acos(cosangle) / Math.PI; // normalized angle
                newpoints[newpoints.length] = new Point(points[i].X, points[i].Y,
                 ↪  points[i].Z, points[i].ID, angle);
        }
        newpoints[newpoints.length] = new Point( // last point
                points[points.length - 1].X,
        points[points.length - 1].Y,
        points[points.length - 1].Z,
                points[points.length - 1].ID);
        return newpoints;
}
function Centroid(points)
{
        var x = 0.0, y = 0.0, z = 0.0;
        for (var i = 0; i < points.length; i++) {
                x += points[i].X;
        y += points[i].Y;
        z += points[i].Z;
        }
        x /= points.length;
    y /= points.length;
    z /= points.length;
        return new Point(x, y, z, 0);
}
function PathLength(points) // length traversed by a point path
{
        var d = 0.0;
        for (var i = 1; i < points.length; i++) {
                if (points[i].ID == points[i-1].ID)
                        d += Distance(points[i-1], points[i]);
        }
        return d;
}
function DistanceWithAngle(p1, p2) // $P+
{
        var dx = p2.X - p1.X;
    var dy = p2.Y - p1.Y;
    var dz = p2.Z - p1.Z;
        var da = p2.Angle - p1.Angle;
        return Math.sqrt(dx * dx + dy * dy + dz * dz + da * da);
}
function Distance(p1, p2) // Euclidean distance between two points
```

```
{
        var dx = p2.X - p1.X;
        var dy = p2.Y - p1.Y;
        var dz = p2.Z - p1.Z;
        return Math.sqrt(dx * dx + dy * dy + dz * dz);
}
function DirDist(points1, points2)
{
        const padding = 2;
        let dist = 0;
        for (var i = padding; i < NumPoints - (1 + padding); i+=1) {
                let v1x = points1[i+1].X - points1[i].X;
                let v1y = points1[i+1].Y - points1[i].Y;
                let v1z = points1[i+1].Z - points1[i].Z;
                let v2x = points2[i+1].X - points2[i].X;
                let v2y = points2[i+1].Y - points2[i].Y;
                let v2z = points2[i+1].Z - points2[i].Z;
                dist += v1x * v2x + v1y * v2y + v1z * v2z;
        }
        return dist;
}

module.exports = {
        Point,
        P3DollarPlusXRecognizer
};
```

recognizer.js

```
const AbstractRecognizer =
 ↪  require('../../../framework/recognizers/abstract-recognizer').AbstractRecognizer;
const P3DollarPlusXRecognizer =
 ↪  require('./p3dollarplusx/p3dollarplusx').P3DollarPlusXRecognizer;
const Point = require('./p3dollarplusx/p3dollarplusx').Point;

class Recognizer extends AbstractRecognizer {

        static name = "P3DollarPlusXRecognizer";

    constructor(options, dataset) {
                super();
                this.samplingPoints = options.samplingPoints;
                this.articulations = options.articulations;
        this.recognizer = new P3DollarPlusXRecognizer(this.samplingPoints);
                if (dataset !== undefined){
                        dataset.getGestureClasses().forEach((gesture) => {
                                gesture.getSamples().forEach(sample => {
                                                this.addGesture(gesture.name,
                                                 ↪  sample);
                                        }
                                );
```

```
                        });
                }
        }

        addGesture(name, sample) {
                let points = convert(sample, this.articulations);
                this.recognizer.AddGesture(name, points);
        }

    removeGesture(name) {
                this.recognizer.RemoveGesture(name);
        }

    recognize(sample) {
                let points = convert(sample, this.articulations);
                if(points.length < 2) {
            return { name: "", time: 0, score: 0.0 };
        }
        let result = this.recognizer.Recognize(points);
                return (result.Name === "No match.") ? { name: "", time:
                 ↪  result.Time, score: result.Score } : { name: result.Name, time:
                 ↪  result.Time, score: result.Score };
        }

        toString() {
        return `${Recognizer.name} [ samplingPoints = ${this.samplingPoints},
         ↪  articulations = ${this.articulations} ]`;
    }

}

function convert(sample, articulations) {
        let points = [];
        articulations.forEach((articulation, articulationID) => {
                sample.paths[articulation].strokes.forEach((stroke, strokeId) => {
                        stroke.points.forEach((point) => {
                                // If multipoint, one stroke per articulation,
                                 ↪  otherwise, keep original strokes
                                let index = articulations.length > 1 ?
                                 ↪  articulationID : strokeId;
                                points.push(new Point(point.x, point.y, point.z,
                                 ↪  index));
                        });
                });
        });
        return points;
}

module.exports = {
        Recognizer
};
```

F.7.5 Q^3

q3dollar.js

```
//
// Point class
//
function Point(x, y, z, id) // constructor
{
        this.X = x;
    this.Y = y;
    this.Z = z;
        this.ID = id;  // stroke ID to which this point belongs (1,2,3,etc.)
        this.IntX = 0; // for indexing into the LUT
    this.IntY = 0; // for indexing into the LUT
    this.IntZ = 0; // for indexing into the LUT
}
//
// PointCloud class
//
function PointCloud(name, points) // constructor
{
        this.Name = name;
        this.Points = Resample(points, NumPoints);
        this.Points = Scale(this.Points);
        this.Points = TranslateTo(this.Points, Origin);
        this.Points = MakeIntCoords(this.Points); // fills in (IntX, IntY, IntZ)
         ↪  values
        this.LUT = ComputeLUT(this.Points);
}
//
// Result class
//
function Result(name, score, ms) // constructor
{
        this.Name = name;
        this.Score = score;
        this.Time = ms;
}
//
// Q3DollarRecognizer constants
//
let NumPoints = 32;
const Origin = new Point(0, 0, 0, 0);
const MaxIntCoord = 1024; // (IntX, IntY, IntZ) range from [0, MaxIntCoord - 1]
const LUTSize = 16; // default size of the lookup table is 16 x 16 x 16
const LUTScaleFactor = MaxIntCoord / LUTSize; // used to scale from (IntX, IntY,
 ↪  IntZ) to LUT
//
```

```
// Q3DollarRecognizer class
//
function Q3DollarRecognizer(numPoints) // constructor
{
    this.PointClouds = new Array();
        NumPoints = numPoints;
        //
        // The $Q3 Point-Cloud Recognizer API begins here -- 4 methods:
        ↪ Recognize(), AddGesture(), RemoveGesture(), DeleteUserGestures()
        //
        this.Recognize = function(points)
        {
                var t0 = Date.now();
                var candidate = new PointCloud("", points);

                var u = -1;
                var b = +Infinity;
                for (var i = 0; i < this.PointClouds.length; i++) // for each
                ↪ point-cloud template
                {
                        var d = CloudMatch(candidate, this.PointClouds[i], b);
                        if (d < b) {
                                b = d; // best (least) distance
                                u = i; // point-cloud index
                        }
                }
                var t1 = Date.now();
                return (u == -1) ? new Result("No match.", 0.0, t1-t0) : new
                ↪ Result(this.PointClouds[u].Name, b > 1.0 ? 1.0 / b : 1.0,
                ↪ t1-t0);
        }
        this.AddGesture = function(name, points)
        {
                this.PointClouds.push(new PointCloud(name, points));
                var num = 0;
                for (var i = 0; i < this.PointClouds.length; i++) {
                        if (this.PointClouds[i].Name == name)
                                num++;
                }
                return num;
    }
    this.RemoveGesture = function(name)
    {
                this.PointClouds = this.PointClouds.filter(pointCloud =>
                ↪ pointCloud.Name !== name);
        }
        this.DeleteUserGestures = function()
        {
                this.PointClouds.length = NumPointClouds; // clears any beyond the
                ↪ original set
                return NumPointClouds;
```

```
89          }
90  }
91  //
92  // Private helper functions from here on down
93  //
94  function CloudMatch(candidate, template, minSoFar)
95  {
96          var n = candidate.Points.length;
97          var step = Math.floor(Math.pow(n, 0.5));
98
99          var LB1 = ComputeLowerBound(candidate.Points, template.Points, step,
            ↪  template.LUT);
100         let LB2 = ComputeLowerBound(template.Points, candidate.Points, step,
            ↪  candidate.LUT);
101
102         for (var i = 0, j = 0; i < n; i += step, j++) {
103                 if (LB1[j] < minSoFar)
104                         minSoFar = Math.min(minSoFar,
                            ↪  CloudDistance(candidate.Points, template.Points, i,
                            ↪  minSoFar));
105                 if (LB2[j] < minSoFar)
106                         minSoFar = Math.min(minSoFar,
                            ↪  CloudDistance(template.Points, candidate.Points, i,
                            ↪  minSoFar));
107         }
108         return minSoFar;
109 }
110 function CloudDistance(pts1, pts2, start, minSoFar)
111 {
112         var n = pts1.length;
113         var unmatched = new Array(); // indices for pts2 that are not matched
114         for (var j = 0; j < n; j++)
115                 unmatched[j] = j;
116         var i = start;  // start matching with point 'start' from pts1
117         var weight = n; // weights decrease from n to 1
118         var sum = 0.0;  // sum distance between the two clouds
119         do
120         {
121                 var u = -1;
122                 var b = +Infinity;
123                 for (var j = 0; j < unmatched.length; j++)
124                 {
125                         d = SqrEuclideanDistance(pts1[i], pts2[unmatched[j]]);
126                         if (d < b) {
127                                 b = d;
128                                 u = j;
129                         }
130                 }
131                 unmatched.splice(u, 1); // remove item at index 'u'
132                 sum += weight * b;
133                 if (sum >= minSoFar)
```

```
                        return sum; // early abandoning
                weight--;
                i = (i + 1) % n;
        } while (i != start);
        return sum;
}
function ComputeLowerBound(pts1, pts2, step, LUT)
{
        var n = pts1.length;
        var LB = new Array(Math.floor(n / step) + 1);
        var SAT = new Array(n);
        LB[0] = 0.0;
        for (var i = 0; i < n; i++)
        {
                var x = Math.round(pts1[i].IntX / LUTScaleFactor);
        var y = Math.round(pts1[i].IntY / LUTScaleFactor);
        var z = Math.round(pts1[i].IntZ / LUTScaleFactor);
                var index = LUT[x][y][z];
                var d = SqrEuclideanDistance(pts1[i], pts2[index]);
                SAT[i] = (i == 0) ? d : SAT[i - 1] + d;
                LB[0] += (n - i) * d;
        }
        for (var i = step, j = 1; i < n; i += step, j++)
                LB[j] = LB[0] + i * SAT[n-1] - n * SAT[i-1];
        return LB;
}
function Resample(points, n)
{
        var I = PathLength(points) / (n - 1); // interval length
        var D = 0.0;
        var newpoints = new Array(points[0]);
        for (var i = 1; i < points.length; i++)
        {
                if (points[i].ID == points[i-1].ID)
                {
                        var d = EuclideanDistance(points[i-1], points[i]);
                        if ((D + d) >= I)
                        {
                                var qx = points[i-1].X + ((I - D) / d) *
                                 ↪  (points[i].X - points[i-1].X);
                                var qy = points[i-1].Y + ((I - D) / d) *
                                 ↪  (points[i].Y - points[i-1].Y);
                                var qz = points[i-1].Z + ((I - D) / d) *
                                 ↪  (points[i].Z - points[i-1].Z);
                                var q = new Point(qx, qy, qz, points[i].ID);
                                newpoints[newpoints.length] = q; // append new
                                 ↪  point 'q'
                                points.splice(i, 0, q); // insert 'q' at position i
                                 ↪  in points s.t. 'q' will be the next i
                                D = 0.0;
                        }
```

```
                        else D += d;
                }
        }
        if (newpoints.length == n - 1) // sometimes we fall a rounding-error short
         ↪  of adding the last point, so add it if so
                newpoints[newpoints.length] = new Point(points[points.length -
                 ↪  1].X, points[points.length - 1].Y, points[points.length - 1].Z,
                 ↪  points[points.length - 1].ID);
        return newpoints;
}
function Scale(points)
{
        var minX = +Infinity, maxX = -Infinity, minY = +Infinity, maxY = -Infinity,
         ↪  minZ = +Infinity, maxZ = -Infinity;
        for (var i = 0; i < points.length; i++) {
                minX = Math.min(minX, points[i].X);
        minY = Math.min(minY, points[i].Y);
        minZ = Math.min(minZ, points[i].Z);
                maxX = Math.max(maxX, points[i].X);
        maxY = Math.max(maxY, points[i].Y);
        maxZ = Math.max(maxZ, points[i].Z);
        }
        var size = Math.max(maxX - minX, maxY - minY, maxZ - minZ);
        var newpoints = new Array();
        for (var i = 0; i < points.length; i++) {
                var qx = (points[i].X - minX) / size;
                var qy = (points[i].Y - minY) / size;
                var qz = (points[i].Z - minZ) / size;
                newpoints[newpoints.length] = new Point(qx, qy, qz, points[i].ID);
        }
        return newpoints;
}
function TranslateTo(points, pt) // translates points' centroid to pt
{
        var c = Centroid(points);
        var newpoints = new Array();
        for (var i = 0; i < points.length; i++) {
                var qx = points[i].X + pt.X - c.X;
                var qy = points[i].Y + pt.Y - c.Y;
                var qz = points[i].Z + pt.Z - c.Z;
                newpoints[newpoints.length] = new Point(qx, qy, qz, points[i].ID);
        }
        return newpoints;
}
function Centroid(points)
{
        var x = 0.0, y = 0.0, z = 0.0;
        for (var i = 0; i < points.length; i++) {
                x += points[i].X;
        y += points[i].Y;
        z += points[i].Z;
```

```
227         }
228         x /= points.length;
229     y /= points.length;
230     z /= points.length;
231         return new Point(x, y, z, 0);
232 }
233 function PathLength(points) // length traversed by a point path
234 {
235         var d = 0.0;
236         for (var i = 1; i < points.length; i++) {
237                 if (points[i].ID == points[i-1].ID)
238                         d += EuclideanDistance(points[i-1], points[i]);
239         }
240         return d;
241 }
242 function MakeIntCoords(points)
243 {
244         for (var i = 0; i < points.length; i++) {
245                 points[i].IntX = Math.round((points[i].X + 1.0) / 2.0 *
                    ↪  (MaxIntCoord - 1));
246                 points[i].IntY = Math.round((points[i].Y + 1.0) / 2.0 *
                    ↪  (MaxIntCoord - 1));
247         points[i].IntZ = Math.round((points[i].Z + 1.0) / 2.0 * (MaxIntCoord - 1));
248     }
249         return points;
250 }
251 function ComputeLUT(points)
252 {
253         var LUT = new Array();
254         for (var i = 0; i < LUTSize; i++) {
255                 LUT[i] = new Array();
256                 for (var j = 0; j < LUTSize; j++) {
257                         LUT[i][j] = new Array();
258                 }
259         }
260
261         for (var x = 0; x < LUTSize; x++)
262         {
263                 for (var y = 0; y < LUTSize; y++)
264                 {
265             for (var z = 0; z < LUTSize; z++)
266             {
267                 var u = -1;
268                 var b = +Infinity;
269                 for (var i = 0; i < points.length; i++)
270                 {
271                     var row = Math.round(points[i].IntX / LUTScaleFactor);
272                     var col = Math.round(points[i].IntY / LUTScaleFactor);
273                     var layer = Math.round(points[i].IntZ / LUTScaleFactor);
274                     var d = ((row - x) * (row - x)) + ((col - y) * (col - y)) +
                        ↪  ((layer - z) * (layer - z));
```

```
                    if (d < b) {
                        b = d;
                        u = i;
                    }
                }
                LUT[x][y][z] = u;
            }
            }
        }
          return LUT;
}
function SqrEuclideanDistance(pt1, pt2)
{
        var dx = pt2.X - pt1.X;
        var dy = pt2.Y - pt1.Y;
        var dz = pt2.Z - pt1.Z;
        return (dx * dx + dy * dy + dz * dz);
}
function EuclideanDistance(pt1, pt2)
{
        var s = SqrEuclideanDistance(pt1, pt2);
        return Math.sqrt(s);
}

module.exports = {
    Q3DollarRecognizer,
    Point
}
```

recognizer.js

```
const AbstractRecognizer =
 ↪  require('../../../framework/recognizers/abstract-recognizer').AbstractRecognizer;
const Q3DollarRecognizer = require('./q3dollar/q3dollar').Q3DollarRecognizer;
const Point = require('./q3dollar/q3dollar').Point;

class Recognizer extends AbstractRecognizer {

        static name = "Q3DollarRecognizer";

    constructor(options, dataset) {
                super();
                this.samplingPoints = options.samplingPoints;
        this.articulations = options.articulations;
        this.recognizer = new Q3DollarRecognizer(this.samplingPoints);
                if (dataset !== undefined){
                        dataset.getGestureClasses().forEach((gesture) => {
                                gesture.getSamples().forEach(sample => {
                                        this.addGesture(gesture.name,
                                         ↪  sample);
                                }
```

```
                                );
                        });
                }
        }

        addGesture(name, sample) {
                let points = convert(sample, this.articulations);
                this.recognizer.AddGesture(name, points);
        }

    removeGesture(name) {
                this.recognizer.RemoveGesture(name);
        }

    recognize(sample) {
                let points = convert(sample, this.articulations);
                if(points.length === 0) {
            return { name: "", time: 0, score: 0.0 };
        }
        let result = this.recognizer.Recognize(points);
                return (result.Name === "No match.") ? { name: "", time:
                 ↪  result.Time, score: result.Score } : { name: result.Name, time:
                 ↪  result.Time, score: result.Score };
        }

        toString() {
        return `${Recognizer.name} [ samplingPoints = ${this.samplingPoints},
         ↪  articulations = ${this.articulations} ]`;
    }

}

function convert(sample, articulations) {
        let points = [];
        articulations.forEach((articulation, articulationID) => {
                sample.paths[articulation].strokes.forEach((stroke, strokeId) => {
                        stroke.points.forEach((point) => {
                                // If multipoint, one stroke per articulation,
                                 ↪  otherwise, keep original strokes
                                let index = articulations.length > 1 ?
                                 ↪  articulationID : strokeId;
                                points.push(new Point(point.x, point.y, point.z,
                                 ↪  index));
                        });
                });
        });
        return points;
}


module.exports = {
```

```
        Recognizer
};
```

F.7.6 Jackknife

The official JavaScript source code of Jackknife is available on GitHub[1].

recognizer.js

```
const AbstractRecognizer =
 ↪  require('../../../framework/recognizers/abstract-recognizer').AbstractRecognizer;
const jackknife_blades = require('./jackknife/jackknife').jackknife_blades;
const Jackknife = require('./jackknife/jackknife_recognizer').Jackknife;
const Vector = require('./jackknife/vector').Vector;
const Sample = require('./jackknife/sample').Sample;

class Recognizer extends AbstractRecognizer {

    static name = "JackknifeRecognizer";

    constructor(options, dataset) {
        super();
        this.N = options.samplingPoints;
        this.articulations = options.articulations;
        let blades = new jackknife_blades();
        blades.set_ip_defaults();
        this.jackknifeRecognizer = new Jackknife(blades)
                if (dataset !== undefined){
                        dataset.getGestureClasses().forEach((gesture) => {
                                gesture.getSamples().forEach(sample => {
                                                this.addGesture(gesture.name,
                                                 ↪  sample, false);
                                        }
                                );
            });
            this.jackknifeRecognizer.train(6, 2, 1.0);
        }
    }

        addGesture(name, sample, train = false) {
        let jackknifeSample = convert(sample, this.articulations, name);
        if (jackknifeSample) {
            this.jackknifeRecognizer.add_template(jackknifeSample);
            if (train) {
                this.jackknifeRecognizer.train(6, 2, 1.0);
            }
        }
    }
```

[1] https://github.com/ISUE/Jackknife

```
38
39      removeGesture(name) {
40          console.log("Cannot remove gesture!");
41      }
42
43      recognize(sample) {
44          let jackknifeSample = convert(sample, this.articulations);
45          if (!jackknifeSample) {
46              return { name: "", time: 0.0 };
47          }
48          let t0 = Date.now();
49          let ret = this.jackknifeRecognizer.classify(jackknifeSample);
50          let t1 = Date.now();
51                  return (ret == -1) ? { name: "", time: t1-t0 } : { name: ret, time:
                     ↪  t1-t0 };
52      }
53
54      toString() {
55          return `${Recognizer.name} [ samplingPoints = ${this.N} ]`;
56      }
57  }
58
59  function convert(sample, articulations, name) {
60      let jackknifeSample;
61      if (name) {
62          jackknifeSample = new Sample(0, name);
63      } else {
64          jackknifeSample = new Sample();
65      }
66      // check min distance START
67      let maxMovement = 0;
68      let threshold = 40;
69      let initPoints = {};
70      for (const articulation of articulations) {
71          initPoints[articulation] = sample.paths[articulation].strokes[0].points[0];
72      }
73      // check min distance END
74      let nFrames = sample.paths[articulations[0]].strokes[0].points.length;
75      let trajectory = [];
76      for (let i = 0; i < nFrames; i++) {
77          let vCoordinates = [];
78          for (const articulation of articulations) {
79              let point = sample.paths[articulation].strokes[0].points[i];
80              // check min distance START
81              let articulationMovement = distance(point, initPoints[articulation]);
82              maxMovement = Math.max(maxMovement, articulationMovement);
83              // check min distance END
84              vCoordinates.push(point.x);
85              vCoordinates.push(point.y);
86              vCoordinates.push(point.z);
87          }
```

```
        trajectory.push(new Vector(vCoordinates));
    }
    jackknifeSample.add_trajectory(trajectory);
    return maxMovement > threshold ? jackknifeSample : null;
}

function distance(p1, p2) // Euclidean distance between two points
{
        var dx = p2.x - p1.x;
        var dy = p2.y - p1.y;
        var dz = p2.z - p1.z;
        return Math.sqrt(dx * dx + dy * dy + dz * dz);
}

module.exports = {
        Recognizer
};
```

F.8 Large User Interface

Please note that we only show the files that we have modified. The complete source code is available on my GitHub repository[2].

F.8.1 Welcome Screen

Intro.jsx

```
import React, { Component } from 'react';
import backDrop from './backdrop.png';
import './Intro.css';
import { css } from 'glamor';
import glamorous from 'glamorous'
import { withStyles } from '@material-ui/core/styles';
import Particles from 'react-particles-js';
import { Redirect } from 'react-router';
import Button from '@material-ui/core/Button';
import Home from '@material-ui/icons/Home';
import Leap from './leap.js';
import LeapMotion from 'leapjs';
// import Leap from './leap.js';
//firebase
import * as firebase from "firebase/app";
import "firebase/database";
const firebaseConfig = {
  apiKey: "AIzaSyDjM37_DSv2RvPQzl5YiVzmgRHfpd4rJFU",
  authDomain: "lui-medialab.firebaseapp.com",
  databaseURL: "https://lui-medialab.firebaseio.com",
  projectId: "lui-medialab",
  storageBucket: "lui-medialab.appspot.com",
  messagingSenderId: "247289397118",
  appId: "1:247289397118:web:eb2bcb0076d4bb4d"
};

if (!firebase.apps.length) {
firebase.initializeApp(firebaseConfig);
}
var database = firebase.database();
var currentRef = database.ref('voice');
//end

const particleOpt = require('./particles.json');

const fadeIn = css.keyframes({
  '0%': { opacity: 0 },
  '100%': { opacity: 1 }
})

```

[2] https://github.com/sluyters/LUI

```
const slideOut = css.keyframes({
  '100%': { transform: 'translateX(-100%)' },
})

const fingers = ["#9bcfed", "#B2EBF2", "#80DEEA", "#4DD0E1", "#26C6DA"];


const styles = {

  backDrop: {
    width: '100%',
    height: '100%',
  },

  canvas: {
    position: 'absolute',
    height: '100%',
    width: '100%',
    zIndex: 0,
    pointerEvents: 'none',
  },

  // "::selection": {
  //   background: "rgba(0,0,0,0)"
  // },

  button: {
    position: 'fixed',
    bottom: '10px',
    left: '10px',
    zIndex: 10,
    color: "rgba(255,255,255,0.3)",
    backgroundColor: "rgba(255,255,255,0.1)"
  },
};

//Animations for entering/exiting the page:
const Wrapper = glamorous.div(props => ({
  animation: props.isMounted ? `${fadeIn} 1s` : props.page === "app" ? '' :
  ↪  `${fadeIn} 4s`,
  position: 'absolute',
  top: '0px',
  left: '0px',
  width: '100vw',
  height: '100vh',
  zIndex: 5
}))

class Intro extends Component {
  constructor(props) {
    super(props);
```

```
91
92      this.state = {
93        exit: false,
94      };
95    }
96
97     componentDidMount() {
98       //google home
99      currentRef.update({"current":"landing"});
100     var something = this;
101     currentRef.on('value', function(snapshot) {
102       console.log(snapshot.val());
103       var db = snapshot.val();
104       var name = db.goto;
105       if (db.update){
106         if (name === "home") {
107             something.setState({ exit: true });
108             currentRef.update({"update":false});
109         }
110       }
111     });
112   }
113
114   handleClick = () => {
115     console.log("clicked")
116   }
117   handleSwipeUp = () => {
118     let { clicked } = this.state;
119     if (clicked != -1) {
120       this.setState({ clicked: -1 });
121       this.getPhotos();
122     } else {
123       this.setState({ exit: true });
124     }
125   }
126   handleExit = () => {
127     this.setState({
128       exit: true
129     })
130   }
131
132   render() {
133     const { classes } = this.props;
134     // Handling redirecting to home screen:
135     if (this.state.exit) {
136       console.log("EXITING")
137       return <Redirect to={{ pathname: "/Home", state: {page: "home"} }} />
138     }
139     if (this.props.page == "intro") {
140       return (
141         <Wrapper isMounted={this.props.isMounted} page={this.props.page}>
```

```
          <div>

            {/* <canvas className={classes.canvas} ref="canvas"></canvas> */}
            <Leap
              handleClick={this.handleClick}
              handleSwipeUp={this.handleSwipeUp}
              handleExit={this.handleExit}
            />

            <div className={classes.backDrop} >
              {/* <input ref={input => this.inputElement = input} > */}

              {/* Component that renders the interactive particles.js: */}
                <Particles
                  params={particleOpt}
                  style={{ zIndex: 5, position: 'absolute' }}
                />

              {/* </input> */}
            </div>

            {/* Rendering the backdrop: */}
            <div >
              <img className={classes.backDrop} src={backDrop} style={{ zIndex: 1,
                ↪  position: 'absolute' }} />
            </div>
          </div>

          {/* Home button: */}
          <Button onClick={() => this.handleExit()}  className={classes.button}>
              <Home/>
          </Button>

        </Wrapper>
      );
    } else {
      return null;
    }
  }

}

export default withStyles(styles)(Intro);
```

leap.js

```
import PropTypes from 'prop-types';
import React from 'react';
import ReactDOM from 'react-dom';
import { withStyles } from '@material-ui/core/styles';
import GestureHandler from '../../gesture-interface/app-interface';
```

```
import { Fade } from '@material-ui/core';

const fingers = ["#9bcfed", "#B2EBF2", "#80DEEA", "#4DD0E1", "#26C6DA"];
const paused_fingers = ["#9bed9b", "#b1f0b1", "#80ea80", "#4ce14c", "#25da25"];

const styles = {
    canvas: {
        position: 'absolute',
        height: '100%',
        width: '100%',
        zIndex: 100,
        pointerEvents: 'none'
    }
};

class Leap extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            frame: {},
            hand: "",
            indexFinger: "",
        }
    }

    componentDidMount() {
        this.gestureHandler = new GestureHandler();
        this.gestureHandler.onEachGesture(function (gesture) {
            console.log(gesture);
            this.setState({ pause: 4 });
        }.bind(this));
        this.gestureHandler.onGesture("rhand_uswipe", function () {
            this.props.handleExit();
        }.bind(this));
        this.gestureHandler.onGesture("rindex_airtap", function () {
            this.props.handleClick();
        }.bind(this));
        this.gestureHandler.onFrame(function (frame) {
            this.traceFingers(frame.fingers);
        }.bind(this));
        this.timer = setInterval(() => {
            if (this.state.pause > 0) {
                this.setState({ pause: this.state.pause - 1 });
            }
        }, 100);
    }

    componentWillUnmount() {
        console.log("Intro leap is unmounted")
        clearInterval(this.timer);
        this.gestureHandler.disconnect();
```

```
    }

    traceFingers(pointables) {
        try {
            // TODO: make canvas and ctx global
            const canvas = this.refs.canvas;
            canvas.width = canvas.clientWidth;
            canvas.height = canvas.clientHeight;
            const ctx = canvas.getContext("2d");
            ctx.clearRect(0, 0, canvas.width, canvas.height);
            const { pause } = this.state;
            pointables.forEach((pointable) => {
                const color = pause > 0 ? paused_fingers[pointable.type] :
                 ↪  fingers[pointable.type];
                const normalized = pointable.normalizedPosition;
                const x = ctx.canvas.width * normalized[0];
                const y = ctx.canvas.height * (1 - normalized[1]);
                const radius = Math.min(20 / Math.abs(pointable.touchDistance),
                 ↪  50);
                this.drawCircle([x, y], radius, color, pointable.type === 1);

                if (pointable.type == 1) {
                    this.setState({
                        indexFinger: { x, y, vel: pointable.tipVelocity[2] }
                    })
                }
            });
        } catch (err) {
            // console.log("ERR", err);
        }
    }

    drawCircle(center, radius, color, fill) {
        const canvas = this.refs.canvas;
        const ctx = canvas.getContext("2d");
        ctx.beginPath();
        ctx.arc(center[0], center[1], radius, 0, 2 * Math.PI);
        ctx.closePath();
        ctx.lineWidth = 10;
        if (fill) {
            ctx.fillStyle = color;
            ctx.fill();
        } else {
            ctx.strokeStyle = color;
            ctx.stroke();
        }
    }

    render() {
        const { classes } = this.props;

```

```
        return (
            <canvas className={classes.canvas} ref="canvas"></canvas>
        )
    }
}


Leap.propTypes = {
    handleUnlock: PropTypes.func
};

// TODO: better default values
Leap.defaultProps = {

};

export default withStyles(styles)(Leap);
```

F.8.2 App Menu

leap.js

```
import PropTypes from 'prop-types';
import React from 'react';
import ReactDOM from 'react-dom';
import { withStyles } from '@material-ui/core/styles';
import GestureHandler from './gesture-interface/app-interface' // Added

const fingers = ["#9bcfed", "#B2EBF2", "#80DEEA", "#4DD0E1", "#26C6DA"];
const paused_fingers = ["#9bed9b", "#b1f0b1", "#80ea80", "#4ce14c", "#25da25"];

const styles = {
    canvas: {
        position: 'absolute',
        height: '100%',
        width: '100%',
        zIndex: 10,
        pointerEvents: 'none'
    }

};

class Leap extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            hand: false,
            indexFinger: "",
            hovered: "",
            clicked: "",
```

```
            pause: 4
        }
    }

    componentDidMount() {
        this.gestureHandler = new GestureHandler();
        this.gestureHandler.onEachGesture(function (gesture) {
            console.log(gesture);
            this.setState({ pause: 4 });
        }.bind(this));
        this.gestureHandler.onGesture("rhand_uswipe", function () {
            this.props.handleSwipeUp();
        }.bind(this));
        this.gestureHandler.onGesture("rindex_airtap", function () {
            var { hovered } = this.state;
            this.setState({ clicked: hovered })
            this.props.handleClick(hovered);
        }.bind(this));
        this.gestureHandler.onFrame(function (frame) {
            if (frame.fingers.length != 0) {
                this.setState({ hand: true });
            } else {
                this.setState({ hand: false });
            }
            this.traceFingers(frame.fingers);
        }.bind(this));
        this.timer = setInterval(() => {
            if (this.state.pause > 0) {
                this.setState({ pause: this.state.pause - 1 });
            }
            if (this.state.hand) {
                var { hovered } = this.state;
                // hovering
                hovered = this.checkHover();
                this.setState({ hovered });
                this.props.handleHover(hovered);
            }
        }, 100);
    }

    componentWillUnmount() {
        clearInterval(this.timer);
        this.gestureHandler.disconnect();
    }

    traceFingers(pointables) {
        try {
            // TODO: make canvas and ctx global
            const canvas = this.refs.canvas;
            canvas.width = canvas.clientWidth;
            canvas.height = canvas.clientHeight;
```

```
            const ctx = canvas.getContext("2d");
            ctx.clearRect(0, 0, canvas.width, canvas.height);

            const { pause } = this.state;

            pointables.forEach((pointable) => {
                const color = pause > 0 ? paused_fingers[pointable.type] :
                 ↪  fingers[pointable.type];
                const normalized = pointable.normalizedPosition;
                const x = ctx.canvas.width * normalized[0];
                const y = ctx.canvas.height * (1 - normalized[1]);
                const radius = Math.min(20 / Math.abs(pointable.touchDistance),
                 ↪  50);
                this.drawCircle([x, y], radius, color, pointable.type === 1);

                if (pointable.type == 1) {
                    this.setState({
                        indexFinger: { x, y, vel: pointable.tipVelocity[2] }
                    })
                }
            });
        } catch (err) {
            // console.log("ERR", err);
        }
    }

    drawCircle(center, radius, color, fill) {
        const canvas = this.refs.canvas;
        const ctx = canvas.getContext("2d");
        ctx.beginPath();
        ctx.arc(center[0], center[1], radius, 0, 2 * Math.PI);
        ctx.closePath();
        ctx.lineWidth = 10;
        if (fill) {
            ctx.fillStyle = color;
            ctx.fill();
        } else {
            ctx.strokeStyle = color;
            ctx.stroke();
        }
    }

    checkHover() {
        // calculate location of cards
        const cards = this.props.cards;
        const { x, y } = this.state.indexFinger;
        for (let i = 0; i < cards.length; i++) {
            if (cards[i]) {
                const dims =
                 ↪  ReactDOM.findDOMNode(cards[i]).getBoundingClientRect();
                if (x > dims.left && x < dims.right &&
```

```
                    y > dims.top && y < dims.bottom) {
                    return ("card" + String(i + 1));
                }
            }
        }
        return ("");
    }

    render() {
        const { classes } = this.props;
        return (
            <canvas className={classes.canvas} ref="canvas"></canvas>
        )
    }
}


Leap.propTypes = {
    cards: PropTypes.array,
    handleHover: PropTypes.func,
    handleClick: PropTypes.func,
    handleExit: PropTypes.func
};

// TODO: better default values
Leap.defaultProps = {

};

export default withStyles(styles)(Leap);
```

F.8.3 Videos App

VideosApp.jsx

```
import React, { Component } from 'react';
import { Grid } from '@material-ui/core';
import { withStyles } from '@material-ui/core/styles';
import classNames from 'classnames';
import YouTube from 'react-youtube';
import Leap from './leap.js'
import { Redirect } from 'react-router';
import glamorous from 'glamorous'
import { css } from 'glamor';
import SwipeableViews from 'react-swipeable-views';
import MobileStepper from '@material-ui/core/MobileStepper';
import Button from '@material-ui/core/Button';
import KeyboardArrowLeft from '@material-ui/icons/KeyboardArrowLeft';
import KeyboardArrowRight from '@material-ui/icons/KeyboardArrowRight';
import Home from '@material-ui/icons/Home';
```

```
//add firebase
import * as firebase from "firebase/app";
import "firebase/database";
import Clear from '@material-ui/icons/Clear';


const zoomIn = css.keyframes({
  '0%': { transform: 'scale(0.5)' },
  '100%': { transform: 'scale(1)' }
})
//firebase
const firebaseConfig = {
  apiKey: "AIzaSyDjM37_DSv2RvPQzl5YiVzmgRHfpd4rJFU",
  authDomain: "lui-medialab.firebaseapp.com",
  databaseURL: "https://lui-medialab.firebaseio.com",
  projectId: "lui-medialab",
  storageBucket: "lui-medialab.appspot.com",
  messagingSenderId: "247289397118",
  appId: "1:247289397118:web:eb2bcb0076d4bb4d"
};

if (!firebase.apps.length) {
firebase.initializeApp(firebaseConfig);
}
var database = firebase.database();
var currentRef = database.ref('voice');
//end

const styles = {

  gallery: {
    animation: `${zoomIn} 0.5s`,
    maxHeight: '95vh',
  },

  carousel: {
    // width: '90%',
    maxHeight: '95vh',
    padding: '0px',
    margin: '0px',
    overflow: 'hidden',
    backgroundColor: 'rgba(0,0,0,0)',
  },

  row: {
    height: '30%',
    margin: '2%'
  },

  cell: {
    display: 'inline-block',
```

```
    width: '100%',
    height: '100%',
    verticalAlign: 'middle',
    boxSizing: 'border-box',
    margin: '0px',
    position: 'relative',
  },

  container: {
    position: 'absolute',
    top: '0px',
    left: '0px',
    width: '100%',
    height: '100%',
    padding: '0px',
    listStyle: 'none',
    overflow: 'none',
    zIndex: '1',
    backgroundColor: '#ECEFF1',
    overflow: 'hidden',
  },

  frameContainer: {
    display: 'block',
    width: '25vw',
    height: '27vh',
    verticalAlign: 'middle',
    // boxSizing: 'border-box',
    padding: '0px',
    // margin: '1.5vw',
    // transform: 'scale(1)',
    // transition: '200ms',
    // border: '2px solid #37474F',
    boxShadow: '0px 0px 10px 2px #999',
    overflow: 'hidden',
    position: 'relative',
    zIndex: 5
    // pointerEvents: 'none'
  },

  zoomed: {
    width: '100vw',
    height: '93vh',
    // top: '0px',
    // left: '0px',
    // position: 'fixed',
    // zIndex: '100',
  },

  hovered: {
    transform: 'scale(1.15)',
```

```
    transition: '200ms ease-out',
    position: 'relative',
    zIndex: 5,
  },

  stepper: {
    // display: 'auto',
    marginLeft: 'auto',
    marginRight: 'auto',
    width: '8em',
    backgroundColor: 'rgba(0,0,0,0)',
    position: "absolute",
    bottom: "1px",
  },

  dots: {
    margin: 'auto',
  },

  button: {
    position: 'fixed',
    bottom: '10px',
    left: '10px',
    color: "rgba(50,50,50,0.8)",
  },

  xbutton: {
    position: 'fixed',
    bottom: '10px',
    right: '10px',
    color: "rgba(50,50,50,0.8)",
  },

  cover: {
    display: 'block',
    width: '25vw',
    height: '27px',
    // verticalAlign: 'middle',
    // boxSizing: 'border-box',
    padding: '0px',
    // margin: '1.5vw',
    // transform: 'scale(1)',
    // transition: '200ms',
    // border: '2px solid #37474F',
    boxShadow: '0px 0px 10px 2px #999',
    overflow: 'hidden',
    zIndex: 50,
    position: 'absolute',
    pointerEvents: 'none'
  },

```

```
  // row: {
  //   pointerEvents: "none"
  // }

};

const videos = [
    { id: 'rnlCGw-0R8g' },
    { id: 'zw47_q9wbBE' },
    { id: '7m6J8W6Ib4w' },
    { id: 'l7uuTnk69Eo' },
    { id: '6ZfuNTqbHE8' },
    { id: '_8w9rOpV3gc' },
    { id: 'c-SE2Qeqj1g' },
    { id: '-AbaV3nrw6E' },
    { id: 'FTPmnZVgDjQ' },
    { id: 'dQw4w9WgXcQ' },
    { id: 'Xnk4seEHmgw' },
    { id: '5G1C3aBY62E' }
]

const opts = {
  height: '350',
  width: '500',
  playerVars: { // https://developers.google.com/youtube/player_parameters
    autoplay: 0,
    controls: 0
  }
};

const fadeIn = css.keyframes({
  '0%': { opacity: 0 },
  '100%': { opacity: 1 }
})
const slideOut = css.keyframes({
  '100%': { transform: 'translateY(-100%)' },
})
const Wrapper = glamorous.div(props => ({
  animation: props.isMounted ? `${slideOut} 2.5s` : `${fadeIn} 1.5s`,
  position: 'absolute',
  top: '0px',
  left: '0px',
  width: '100vw',
  height: '100vh',
  zIndex: 5
}))


class VideosApp extends Component {
    constructor(props) {
        super(props);
```

```
        this.state = {
            videos: [],
            target_dict: {},
            playing:[],
            zoomed: -1,
            hovered: "",
            index: 0,
            exit: false,
        }
    }

    componentDidMount() {
        console.log("MOUNTED");
        this.getVideos();
        //google home
    currentRef.update({"current":"videos"});
    var something = this;
    currentRef.on('value', function(snapshot) {
      console.log(snapshot.val());
      var db = snapshot.val();
      var name = db.goto;
      if (db.update){
        if (name === "home") {
            something.setState({ exit: true });
            currentRef.update({"update":false});
        }
      }
      if(db.back){
        something.setState({ exit: true });
        currentRef.update({"back":false});
      }

    });
    //end
    }

    getVideos = () => {
        const videos = [this.refs.video1, this.refs.video2, this.refs.video3,
                        this.refs.video4, this.refs.video5, this.refs.video6,
                        this.refs.video7, this.refs.video8, this.refs.video9,
                        this.refs.video10, this.refs.video11, this.refs.video12];
        this.setState({videos});
    }

    handleHover = (video) => {
        this.setState({hovered:video})
    }

    handleSwipe = (dir) => {
      var { zoomed, videos } = this.state;
      // swipe between zoomed in videos
```

```
      if (zoomed != -1) {
        if (dir === "right") {
          zoomed = Math.max(0, zoomed - 1);
        } else {
          zoomed = Math.min(videos.length - 1, zoomed + 1);
        }
        console.log("SWIPE RIGHT (app) " + zoomed)
        this.setState({ zoomed });
      }
      else {
        if (dir === "left") {
          this.setState({
            index: 1
          })
        } else {
          this.setState({
            index: 0
          })
        }
      }
    }

    handleNext = () => {        //for pointer
      this.setState(state => ({
        index: 1,
      }));
    };

    handleBack = () => {
      this.setState(state => ({
        index: 0,
      }));
    };

    handleExit = () => {
      this.setState({
        exit: true
      })
    }

    handleClick = (video) => {
        try{
            var { playing, target_dict } = this.state;
            var videoIndex = parseInt(video.slice(5));
            var videoId = videos[videoIndex-1].id;
            console.log("HANDLE CLICK", videoId)
            if (!playing[videoIndex-1]) {
              playing[videoIndex-1] = true;
              console.log(target_dict[videoId])
              target_dict[videoId].playVideo();
              console.log("PLAY", videoIndex);
```

```
            } else {
              playing[videoIndex-1] = false;
              target_dict[videoId].pauseVideo();
              console.log("PAUSE", videoIndex);
            }
            this.setState({ playing });
        } catch (err) { }
    }

    handleZoom = (video) => {
      var { zoomed } = this.state;
      console.log("ZOOMED", video);
      zoomed = parseInt(video.slice(5)) - 1;
      this.setState({ zoomed, hovered: "" });
    }

    handleIndex = (video, translation) => {
      const threshold = 0.5;
      var videoIndex = parseInt(video.slice(5));
      var videoId = videos[videoIndex-1].id;
      if (Math.abs(translation[0]) > threshold || Math.abs(translation[1]) >
       ↪  threshold) {
        if (Math.abs(translation[0]) > Math.abs(translation[1])) {
          var seconds = this.state.target_dict[videoId].getCurrentTime() -
           ↪  Math.round(translation[0]);
          console.log("HANDLE FAST-FORWARD", videoId, seconds)
          this.state.target_dict[videoId].seekTo(seconds, true);
        } else {
          var volume = Math.round(this.state.target_dict[videoId].getVolume() +
           ↪  translation[1]);
          console.log("HANDLE VOLUME", videoId, volume)
          this.state.target_dict[videoId].setVolume(volume);
        }
      }
    }

    // adjust volume
    handleKnob = (video, volume) => {
      var videoIndex = parseInt(video.slice(5));
      var videoId = videos[videoIndex-1].id;
      console.log("HANDLE KNOB", videoId, volume)
      this.state.target_dict[videoId].setVolume(volume);
    }

    handleSwipeUp = () => {
      let { zoomed } = this.state;
      if (zoomed != -1) {
        this.setState({ zoomed: -1 });
        this.getVideos();
      } else {
        this.setState({ exit: true });
```

```
    }
  }

  _onReady = (videoId, event) => {
    // access to player in all event handlers via event.target
    event.target.pauseVideo();
    this.setState({playing:[...this.state.playing,false]});
    var new_target_dict = this.state.target_dict;
    new_target_dict[videoId] = event.target;
    this.setState({target_dict: new_target_dict});
  }

  getVideoClass(video) {
    const { classes } = this.props;
    const { hovered } = this.state;
    if (hovered === video)
      return classNames(classes.frameContainer, classes.hovered);
    return classes.frameContainer;
  }

  renderVideo(index) {
    const { classes } = this.props;
    const { hovered } = this.state;
    const ref = "video" + String(index + 1);

    return (

      <Grid onMouseEnter={() => { this.setState({hovered: ref}) }}
            onMouseLeave={() => { this.setState({hovered: ""}) }}
            onClick={() => { this.handleClick(hovered) }}
            item
            className={classes.cell}
            xs={12} sm={ 4 }>
        <div className="cover">
          <YouTube ref={ref} item
            className={this.getVideoClass(ref)}
            videoId={videos[index].id}
            key={index}
            opts={opts}
            onReady={(event) => this._onReady(videos[index].id, event)}
            onPause={this._onPause}
          />
        </div>
      </Grid>);
  }

  renderVideos() {
    const { classes } = this.props;

    return (<div>
        <SwipeableViews className={classes.gallery} index={this.state.index}
          ↪  onTransitionEnd={this.getVideos}>
```

```
            <div className={classes.carousel}>
              <Grid container className={classes.row} spacing={0}
               ↪  justify={"center"} >
                {this.renderVideo(0)}
                {this.renderVideo(1)}
                {this.renderVideo(2)}
              </Grid>
              <Grid container className={classes.row} spacing={0}
               ↪  justify={"center"}>
                {this.renderVideo(3)}
                {this.renderVideo(4)}
                {this.renderVideo(5)}
              </Grid>
              <Grid container className={classes.row} spacing={0}
               ↪  justify={"center"} >
                {this.renderVideo(6)}
                {this.renderVideo(7)}
                {this.renderVideo(8)}
              </Grid>
            </div>
            <div className={classes.carousel}>
              <Grid container className={classes.row} spacing={0}
               ↪  justify={"center"} >
                {this.renderVideo(9)}
                {this.renderVideo(10)}
                {this.renderVideo(11)}
              </Grid>
            </div>
          </SwipeableViews>
          <MobileStepper
          variant="dots"
          steps={2}
          position="bottom"
          activeStep={this.state.index}
          className={classes.stepper}
          classes={{ dots: classes.dots }}
          nextButton={
            <Button size="small" onClick={this.handleNext}
             ↪  disabled={this.state.index === 1}>
              <KeyboardArrowRight />
              {/* {theme.direction === 'rtl' ? <KeyboardArrowLeft /> :
               ↪  <KeyboardArrowRight />} */}
            </Button>
          }
          backButton={
            <Button size="small" onClick={this.handleBack}
             ↪  disabled={this.state.index === 0}>
              {/* {theme.direction === 'rtl' ? <KeyboardArrowRight /> :
               ↪  <KeyboardArrowLeft />} */}
              <KeyboardArrowLeft />
            </Button>
```

```
464           }
465         />
466         </div>
467       );
468     }
469
470     renderFullScreenVideo(index) {
471       const { classes } = this.props;
472
473       return (<div className={classes.carousel} justify={"center"}>
474         <Grid container spacing={0} justify={"center"} >
475           <Grid item
476                 className={classes.cell}
477                 xs={12} sm={ 12 }>
478             <YouTube item
479               className={classNames(classes.frameContainer, classes.zoomed)}
480               videoId={videos[index].id}
481               key={index}
482               opts={opts}
483               onReady={(event) => this._onReady(videos[index].id, event)}
484               onPause={this._onPause}
485             />
486           </Grid>
487         </Grid>
488       </div>);
489     }
490
491     renderFullScreen(index) {
492       const { classes } = this.props;
493
494       return (<div>
495         <SwipeableViews className={classes.gallery} index={index}
          ↪  onTransitionEnd={this.getVideos}>
496           { this.renderFullScreenVideo(0) }
497           { this.renderFullScreenVideo(1) }
498           { this.renderFullScreenVideo(2) }
499           { this.renderFullScreenVideo(3) }
500           { this.renderFullScreenVideo(4) }
501           { this.renderFullScreenVideo(5) }
502           { this.renderFullScreenVideo(6) }
503           { this.renderFullScreenVideo(7) }
504           { this.renderFullScreenVideo(8) }
505           { this.renderFullScreenVideo(9) }
506           { this.renderFullScreenVideo(10) }
507           { this.renderFullScreenVideo(11) }
508         </SwipeableViews>
509         <div className = "stepper">
510         <MobileStepper
511           variant="dots"
512           steps={12}
513           position="bottom"
```

```
          activeStep={index}
          className={classes.stepper}
          classes={{ dots: classes.dots }}
          nextButton={
            <Button size="small" onClick={()=>this.handleSwipe("left")}
             ↪  disabled={index === 11}>
              <KeyboardArrowRight />
            </Button>
          }
          backButton={
            <Button size="small" onClick={()=>this.handleSwipe("right")}
             ↪  disabled={index === 0}>
              <KeyboardArrowLeft />
            </Button>
          }
        />
      </div>
      <Button onClick={() => this.handleSwipeUp()}  className={classes.xbutton}>
        <Clear/>
      </Button>
      </div>);
  }

  render() {
      const { classes } = this.props;
      const { zoomed } = this.state;

      if (this.state.exit) {
        return <Redirect to={{ pathname: "/Home", state: {page: "home"} }} />
      }

      return (
        <Wrapper isMounted={this.props.isMounted} exit={this.state.exit}>
          <div className={classes.container}>
            <Leap
              videos={this.state.videos}
              handleHover={this.handleHover}
              handleSwipe={this.handleSwipe}
              handleSwipeUp={this.handleSwipeUp}
              handleClick={this.handleClick}
              handleZoom={this.handleZoom}
              handleKnob={this.handleKnob}
              handleIndex={this.handleIndex}
            />

            { zoomed != -1 ? this.renderFullScreen(zoomed) : this.renderVideos()
             ↪  }

            <Button onClick={() => this.handleExit()}
             ↪  className={classes.button}>
              <Home/>
```

```
              </Button>
            </div>
            </Wrapper>
        );
      }
  }

  VideosApp.defaultProps = {
      hovered: false,
      clicked: false,
  };

export default withStyles(styles)(VideosApp);
```

leap.js

```
import PropTypes from 'prop-types';
import React from 'react';
import ReactDOM from 'react-dom';
import { withStyles } from '@material-ui/core/styles';
//import LeapMotion from 'leapjs';
import GestureHandler from '../../gesture-interface/app-interface' // Added

const fingers = ["#9bcfed", "#B2EBF2", "#80DEEA", "#4DD0E1", "#26C6DA"];
const paused_fingers = ["#9bed9b", "#b1f0b1", "#80ea80", "#4ce14c", "#25da25"];

const videoSizeOffset = 100;

const styles = {
    canvas: {
        position: 'absolute',
        height: '100%',
        width: '100%',
        zIndex: 100,
        pointerEvents: 'none'
    }
};

class Leap extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            indexFinger: "",
            hovered: "",
            zoomed: "",
            pause: 4,
        }
    }

    componentDidMount() {
        let gestureHandler = new GestureHandler();
```

```
        this.gestureHandler = gestureHandler;
        gestureHandler.onEachGesture(function (gesture) {
            console.log(gesture);
            this.setState({ pause: 4 });
        }.bind(this));
        gestureHandler.onGesture("rhand_lswipe", function () {
            this.props.handleSwipe("left");
            let zoomed = this.state.zoomed;
            if (zoomed) {
                const index = parseInt(zoomed.slice(5));
                const newIndex = Math.min(this.props.videos.length, index + 1)
                zoomed = "video" + String(newIndex);
                console.log("SWIPE LEFT (leap) " + zoomed)
                this.setState({ zoomed });
            }
        }.bind(this));
        gestureHandler.onGesture("rhand_rswipe", function () {
            this.props.handleSwipe("right");
            let zoomed = this.state.zoomed;
            if (zoomed) {
                const index = parseInt(zoomed.slice(5));
                const newIndex = Math.max(1, index - 1)
                zoomed = "video" + String(newIndex);
                console.log("SWIPE RIGHT (leap) " + zoomed)
                this.setState({ zoomed });
            }
        }.bind(this));
        gestureHandler.onGesture("rhand_uswipe", function () {
            let zoomed = this.state.zoomed;
            if (zoomed) {
                zoomed = "";
                gestureHandler.removePoseHandler("point-index");
                this.setState({ zoomed });
            }
            this.props.handleSwipeUp();
        }.bind(this));
        gestureHandler.onGesture("rindex_airtap", function () {
            var { zoomed, hovered } = this.state;
            if (hovered && !zoomed) { // zoom in
                zoomed = hovered;
                this.props.handleZoom(zoomed);
                this.setState({ zoomed, hovered: "" });
                // Add gesture handler for volume
                gestureHandler.onPose("point-index", function (data) {
                    let zoomed = this.state.zoomed;
                    if (zoomed) {
                        this.props.handleIndex(zoomed, data.translation);
                    }
                }.bind(this));
            } else if (zoomed) {  // play video
                console.log("PLAY (leap) " + zoomed)
```

```
                this.props.handleClick(zoomed);
            }
        }.bind(this));
        this.gestureHandler.onFrame(function (frame) {
            if (frame.fingers.length != 0) {
                this.setState({ hand: true });
            } else {
                this.setState({ hand: false });
            }
            this.traceFingers(frame.fingers);
        }.bind(this));

        this.timer = setInterval(() => {
            if (this.state.pause > 0) {
                this.setState({ pause: this.state.pause - 1 });
            }
            if (this.state.hand) {
                var { zoomed, hovered } = this.state;
                if (!zoomed) {
                    hovered = this.checkHover();
                    this.setState({ hovered });
                    this.props.handleHover(hovered);
                }
            }
        }, 100);
    }

    componentWillUnmount() {
        clearInterval(this.timer);
        this.gestureHandler.disconnect();
    }

    traceFingers(pointables) {
        try {
            // TODO: make canvas and ctx global
            const canvas = this.refs.canvas;
            canvas.width = canvas.clientWidth;
            canvas.height = canvas.clientHeight;
            const ctx = canvas.getContext("2d");
            ctx.clearRect(0, 0, canvas.width, canvas.height);
            const { pause } = this.state;
            pointables.forEach((pointable) => {
                const color = pause > 0 ? paused_fingers[pointable.type] :
                 ↪  fingers[pointable.type];
                const normalized = pointable.normalizedPosition;
                const x = ctx.canvas.width * normalized[0];
                const y = ctx.canvas.height * (1 - normalized[1]);
                const radius = Math.min(20 / Math.abs(pointable.touchDistance),
                 ↪  50);
                this.drawCircle([x, y], radius, color, pointable.type === 1);

```

```
                if (pointable.type == 1) {
                    this.setState({
                        indexFinger: { x, y, vel: pointable.tipVelocity[2] }
                    })
                }
            });
        } catch (err) {
            // console.log("ERR", err);
        }
    }

    drawCircle(center, radius, color, fill) { //draws a circle
        const canvas = this.refs.canvas;
        const ctx = canvas.getContext("2d");
        ctx.beginPath();
        ctx.arc(center[0], center[1], radius, 0, 2 * Math.PI);
        ctx.closePath();
        ctx.lineWidth = 10;
        if (fill) {
            ctx.fillStyle = color;
            ctx.fill();
        } else {
            ctx.strokeStyle = color;
            ctx.stroke();
        }
    }

    checkHover() {
        const videos = this.props.videos;
        const { x, y } = this.state.indexFinger;
        // don't check for hovering while zoomed in
        if (!this.state.zoomed) {
          for (let i = 0; i < videos.length; i++) {
              if (videos[i]){
                  const videoNode = ReactDOM.findDOMNode(videos[i]);
                  const dims = videoNode.getBoundingClientRect();
                  if (x > dims.left && x < dims.right &&
                      y > dims.top - videoSizeOffset && y < dims.bottom +
                       ↪  videoSizeOffset) {
                      return ("video" + String(i + 1));
                  }
              }
          }
        }
        return ("");
    }

    render() {
        const { classes } = this.props;
        return (
            <canvas className={classes.canvas} ref="canvas"></canvas>
```

```
        )
    }
}


Leap.propTypes = {
    videos: PropTypes.array,
    handleHover: PropTypes.func,
    handleSwipe: PropTypes.func,
    handleExit: PropTypes.func,
    handleClick: PropTypes.func,
    handleIndex: PropTypes.func
};

// TODO: better default values
Leap.defaultProps = {
    videos: [],
};

export default withStyles(styles)(Leap);
```

F.8.4 Photos App

PhotosApp.jsx

```
import React, { Component } from 'react';
import { Redirect } from 'react-router';
import { withStyles } from '@material-ui/core/styles';
import glamorous from 'glamorous'
import classNames from 'classnames';
import Leap from './leap.js';
import { Grid } from '@material-ui/core';
import SwipeableViews from 'react-swipeable-views';
import MobileStepper from '@material-ui/core/MobileStepper';
import Button from '@material-ui/core/Button';
import KeyboardArrowLeft from '@material-ui/icons/KeyboardArrowLeft';
import KeyboardArrowRight from '@material-ui/icons/KeyboardArrowRight';
import Home from '@material-ui/icons/Home';
import Clear from '@material-ui/icons/Clear';
import FavoriteIcon from '@material-ui/icons/Favorite';
import FavoriteBorderIcon from '@material-ui/icons/FavoriteBorder';

import { css } from 'glamor';
import { Transition } from 'react-transition-group';
import Carousel from 'react-responsive-carousel';
//firebase
import * as firebase from "firebase/app";
import "firebase/database";

const zoomIn = css.keyframes({
```

```
  '0%': { transform: 'scale(0.5)' },
  '100%': { transform: 'scale(1)' }
})

//CSS:
const styles = {

  gallery: {
    animation: `${zoomIn} 1s`
  },

  container: {
    display: 'flex',
    flexDirection: 'column',
    position: 'absolute',
    top: '0px',
    left: '0px',
    width: '100%',
    height: '100%',
    padding: '0px',
    listStyle: 'none',
    overflow: 'hidden',
    zIndex: '1',
    backgroundColor: '#CFD8DC',
  },

  carousel: {
    // width: '90%',
    maxHeight: '95%',
    padding: '0px',
    margin: '0px',
    overflow: 'hidden',
  },

  row: {
    maxHeight: '50vh',
  },

  cell: {
    display: 'inline-block',
    maxWidth: '85%',
    maxHeight: '85%',
    verticalAlign: 'middle',
    boxSizing: 'border-box',
    margin: '0px',
    padding: '3%',
    position: 'relative',
  },

  image: {
    display: 'block',
```

```
    maxWidth: '90%',
    maxHeight: '60%',
    width: 'auto',
    height: 'auto',
    margin: 'auto',
    padding: '3%',
    border: 'none',
    transform: 'scale(1)',
    transition: 'all 0.4s',
    boxShadow: '0px 0px 10px 2px #999',
    backgroundColor: "#ECEFF1",
    position: "relative",
    zIndex: '1'
  },

  hovered: {
    transform: 'scale(1.5)',
    animationDuration: '0.1s',
    zIndex: '15 !important',
    cursor: 'pointer',
  },

  zoomed: {
    maxHeight: '80vh'
  },

  dots: {
    margin: 'auto',
  },

  maincontent: {
    position: 'absolute',
    bottom: '50px',
    top: '50px',
    left: 0,
    right: 0,
    overflow: 'hidden'
  },

  controlbar: {
    boxSizing: 'border-box',
    display: 'flex',
    justifyContent: 'space-between',
    alignItems: 'center',
    margin: 0,
    padding: '5px',
    position: 'absolute',
    bottom: 0,
    left: 0,
    right: 0,
    height: '50px'
```

```
  },

  control: {
    flex: '1 1 0'
  },

  stepper: {
    // display: 'auto',
    margin: 'auto',
    width: 'fit-content',
    maxWidth: '100%',
    // marginLeft: 'auto',
    // marginRight: 'auto',
    //width: '8em',
    backgroundColor: 'rgb(0,0,0,0)',
    //position: "absolute",
    //bottom: "1px",
  },

  button: {
    //position: 'fixed',
    // bottom: '10px',
    // left: '10px',
    color: "rgba(50,50,50,0.8)",
  },

  xbutton: {
    position: 'fixed',
    top: '10px',
    right: '10px',
    color: "rgba(50,50,50,0.8)",
  }
};

const fadeIn = css.keyframes({
  '0%': { opacity: 0 },
  '100%': { opacity: 1 }
})
const slideOut = css.keyframes({
  '100%': { transform: 'translateY(-100%)' },
})

//Animations entering/exiting this page:
const Wrapper = glamorous.div(props => ({
  animation: props.isMounted ? `${slideOut} 2.5s` : `${fadeIn} 1.5s`,
  position: 'absolute',
  top: '0px',
  left: '0px',
  width: '100vw',
  height: '100vh',
  zIndex: 5
```

```
179 }))
180 const photos =
    ↪  ['https://images.unsplash.com/photo-1531752074002-abf991376d04?ixlib=rb-0.3.5&ixid=eyJhcH
181
                ↪  'https://images.unsplash.com/photo-1531700968341-bd114e5006ec?ixlib=rb-0.3
182
                ↪  'https://images.unsplash.com/photo-1533247094082-709d7257cb7b?ixlib=rb-0.3
183
                ↪  'https://images.unsplash.com/photo-1531686888376-83ee7d64f5eb?ixlib=rb-0.3
184
                ↪  'https://images.unsplash.com/photo-1533213603451-060f6ec38bfa?ixlib=rb-0.3
185
                ↪  'https://images.unsplash.com/photo-1531574595918-cb77c99fe5e2?ixlib=rb-0.3
186
                ↪  'https://images.unsplash.com/photo-1533135347859-2d07f6e8199b?ixlib=rb-0.3
187
                ↪  'https://images.unsplash.com/photo-1533071581733-1a1600ec8ac6?ixlib=rb-0.3
188
                ↪  'https://images.unsplash.com/photo-1533213603451-060f6ec38bfa?ixlib=rb-0.3
189
                ↪  'https://images.unsplash.com/photo-1531574595918-cb77c99fe5e2?ixlib=rb-0.3
190
                ↪  'https://images.unsplash.com/photo-1533135347859-2d07f6e8199b?ixlib=rb-0.3
191
                ↪  'https://images.unsplash.com/photo-1533071581733-1a1600ec8ac6?ixlib=rb-0.3
192
                ↪  'https://images.unsplash.com/photo-1531752074002-abf991376d04?ixlib=rb-0.3
193
                ↪  'https://images.unsplash.com/photo-1531700968341-bd114e5006ec?ixlib=rb-0.3
194
                ↪  'https://images.unsplash.com/photo-1533247094082-709d7257cb7b?ixlib=rb-0.3
195
                ↪  'https://images.unsplash.com/photo-1531686888376-83ee7d64f5eb?ixlib=rb-0.3
196
                ↪  'https://images.unsplash.com/photo-1531686888376-83ee7d64f5eb?ixlib=rb-0.3
197
                ↪  'https://images.unsplash.com/photo-1531686888376-83ee7d64f5eb?ixlib=rb-0.3
198
                ↪  'https://images.unsplash.com/photo-1531686888376-83ee7d64f5eb?ixlib=rb-0.3
199
                ↪  'https://images.unsplash.com/photo-1531686888376-83ee7d64f5eb?ixlib=rb-0.3
200
201 //firebase
202 const firebaseConfig = {
203   apiKey: "AIzaSyDjM37_DSv2RvPQzl5YiVzmgRHfpd4rJFU",
204   authDomain: "lui-medialab.firebaseapp.com",
205   databaseURL: "https://lui-medialab.firebaseio.com",
206   projectId: "lui-medialab",
207   storageBucket: "lui-medialab.appspot.com",
208   messagingSenderId: "247289397118",
209   appId: "1:247289397118:web:eb2bcb0076d4bb4d"
```

```
};

if (!firebase.apps.length) {
firebase.initializeApp(firebaseConfig);
}
var database = firebase.database();
var currentRef = database.ref('voice');
//end


class PhotosApp extends Component {
  constructor(props) {
    super(props);

    let photoRefs = [];
    for (let i = 0; i < photos.length; i++) {
      photoRefs.push(React.createRef());
    }

    this.state = {
      photoRefs: photoRefs,
      numPreviewPages: Math.ceil(photoRefs.length / 8),
      hovered: -1,
      clicked: -1,
      liked: {},
      index: 0,
      exit: false,
      rotation: 0, // Added - rotate image
      zoom: 1.0,
      translation: { x: 0.0, y: 0.0 }
    };

    this.handleClick = this.handleClick.bind(this);
  }

  componentDidMount() {
    //google home
    currentRef.update({"current":"photos"});
    var something = this;
    currentRef.on('value', function(snapshot) {
      var db = snapshot.val();
      var name = db.goto;
      if (db.update){
        if (name === "home") {
            something.setState({ exit: true });
            currentRef.update({"update":false});
        }

      }
      if(db.clicked){
        currentRef.update({"clicked":false});
```

```
        something.gestureDetected = true;
        console.log(db.hovered);
        something.handleClick(db.hovered);
      }
      if (db.back){
        if (something.state.clicked != -1){
          something.handleSwipeUp();
        }
        else{
          something.setState({ exit: true });

        }
        currentRef.update({"back":false});
      }
    });
  }



  rotate(){
    let newRotation = this.state.rotation + 45;
    if(newRotation >= 360){
      newRotation =- 360;
    }
    this.setState({
      rotation: newRotation,
    })
  }

  handleHover = (photo) => {
    this.setState({ hovered: photo })
    currentRef.update({"hovered": photo});
  }

  handleClick = (photoId) => {
    this.setState({ clicked: photoId, rotation: 0, zoom: 1.0, translation: { x:
     ↪  0.0, y: 0.0 } })
  }

  handleExit = () => {
    this.setState({
      exit: true
    })
  }

  handleRotate = (handRotation) => {
    this.setState(prevState => ({ rotation: (prevState.rotation + 180 *
     ↪  handRotation / 3.14) }))
  }

  handlePinch = (pinch, handTranslation) => {
```

```
    let { clicked, zoom, translation } = this.state;
    if (clicked != -1) {
      zoom *= ((pinch - 1) * 1.2) + 1;
      if (zoom < 0.4) {
        zoom = 0.4;
      } else if (zoom > 4) {
        zoom = 4;
      }
      translation.x += handTranslation[0] * 5;
      translation.y += handTranslation[1] * -5;
      this.setState({ zoom, translation })
    }
  }

  handleTranslate = (translation) => {
    console.log("translate");
    console.log(translation);
  }

  handleThumb = (isUp) => {
    let { clicked, liked } = this.state;

    if (!liked.hasOwnProperty(clicked) || liked[clicked] != isUp) {
      this.setState(prevState => ({
        liked: {
          ...prevState.liked,
          [clicked]: isUp
        }
      }));
    }
  }

  handleSwipe = (dir) => {
    let { clicked, photoRefs, index, numPreviewPages } = this.state;
    this.setState({ rotation: 0, zoom: 1.0, translation: { x: 0.0, y: 0.0 } })
     ↪  // Added - rotation
    if (dir === "left") {
      console.log("swipe left");
      if (clicked != -1) {
        this.setState({ clicked: Math.min(photoRefs.length - 1, clicked + 1) })
      } else if (index < numPreviewPages - 1) {
        this.setState({ index: index + 1 })
      }
    } else {
      console.log("swipe right");
      if (clicked != -1) {
        this.setState({ clicked: Math.max(0, clicked - 1) })
      } else if (index > 0) {
        this.setState({ index: index - 1 })
      }
    }
```

```
  }

  handleNext = () => {
    let { index, numPreviewPages } = this.state;
    if (index < numPreviewPages - 1) {
      this.setState({ index: index + 1 });
    }
  };

  handleBack = () => {
    let { index } = this.state;
    if (index > 0) {
      this.setState({ index: index - 1 });
    }
  };

  handleSwipeUp = () => {
    let { clicked } = this.state;
    if (clicked != -1) {
      this.setState({ clicked: -1 });
    } else {
      this.setState({ exit: true });
    }
  }

  renderPhoto(index) { //renders a photo in the grid
    const { classes } = this.props;
    const { hovered, photoRefs } = this.state;
    const ref = photoRefs[index];

    return (<Grid item className={classes.cell} ref={ref} xs={12} sm={3}>
      <img
        onMouseEnter={() => { this.setState({hovered: index}) }}
        onMouseLeave={() => { this.setState({hovered: -1}) }}
        onClick={() => { this.handleClick(index) }}
        className={hovered === index ? classNames(classes.image, classes.hovered) :
        ↪  classes.image}
        src={ photos[index] } />
    </Grid>);
  }

  renderFullScreenPhoto(index) { //renders the selected photo in full screen view
    const { classes } = this.props;
    const { rotation, zoom, translation } =  this.state;
    return (<div className={classes.carousel} justify={"center"}>
      <Grid container spacing={0} justify={"center"} >
        <Grid item className={classes.cell} xs={12} sm={12}>
          <div style={{transform:
          ↪  `translate(${translation.x}px,${translation.y}px)`}} >
            <img
              style={{transform: `rotate(${rotation}deg) scale(${zoom})`}}
```

```
409             onClick={() => { this.rotate() }}
410             className={classNames(classes.image, classes.zoomed)}
411             src={photos[index]} />
412         </div>
413       </Grid>
414     </Grid>
415   </div>);
416 }
417
418 renderFullScreen(index) { //renders the full screen gallery view for the photos
419   const { classes } = this.props;
420   const { photoRefs } = this.state;
421
422   let fullScreenPhotos = [];
423   for (let i = 0; i < photoRefs.length; i++) {
424     fullScreenPhotos.push(this.renderFullScreenPhoto(i));
425   }
426
427   return (<div>
428     <SwipeableViews className={classes.gallery} index={index} >
429       {fullScreenPhotos}
430     </SwipeableViews>
431     {/* Exit button: */}
432     <Button onClick={() => this.handleSwipeUp()}  className={classes.xbutton}>
433       <Clear/>
434     </Button>
435   </div>);
436 }
437
438 renderPhotos() {
439   const { classes } = this.props;
440   const { index, photoRefs } = this.state;
441   let pages = [];
442   let rows = [];
443   let previews = [];
444   var i;
445   for (i = 0; i < photoRefs.length; i++) {
446     previews.push(this.renderPhoto(i));
447     if ((i + 1) % 4 == 0) {
448       rows.push(<Grid container className={classes.row} spacing={0}
        ↪  justify={"center"} >
449         {previews}
450       </Grid>);
451       previews = [];
452     }
453     if ((i + 1) % 8 == 0) {
454       pages.push(<div className={classes.carousel} justify={"center"}>
455         {rows}
456       </div>);
457       rows = [];
458     }
```

```
    }
    if ((photoRefs.length % 8) != 0) {
      if ((photoRefs.length % 4) != 0) {
        rows.push(<Grid container className={classes.row} spacing={0}
        ↪  justify={"center"} >
          {previews}
        </Grid>);
      }
      pages.push(<div className={classes.carousel} justify={"center"}>
        {rows}
      </div>);
    }

    return (<div>
      <div>
        <SwipeableViews className={classes.gallery} index={index}>
          {pages}
        </SwipeableViews>
      </div>
    </div>);
  }

  renderStepper() {
    const { classes } = this.props;
    const { index, numPreviewPages } = this.state;

    return (numPreviewPages <= 1) ? "" : (<div className = "stepper">
      <MobileStepper
        variant="dots"
        steps={numPreviewPages}
        position="bottom"
        activeStep={index}
        className={classes.stepper}
        classes={{ dots: classes.dots }}
        nextButton={
          <Button size="small" onClick={this.handleNext} disabled={index >=
          ↪  numPreviewPages - 1}>
            <KeyboardArrowRight />
          </Button>
        }
        backButton={
          <Button size="small" onClick={this.handleBack} disabled={index <= 0}>
            <KeyboardArrowLeft />
          </Button>
        }
      />
    </div>);
  }

  renderFullScreenStepper() {
    const { classes } = this.props;
```

```
    const { clicked, photoRefs } = this.state;

    return (<div className = "stepper">
        <MobileStepper
          variant="dots"
          steps={photoRefs.length}
          position="bottom"
          activeStep={clicked}
          className={classes.stepper}
          classes={{ dots: classes.dots }}
          nextButton={
            <Button size="small" onClick={()=>this.handleSwipe("left")}
             ↪  disabled={clicked >= photoRefs.length - 1}>
              <KeyboardArrowRight />
            </Button>
          }
          backButton={
            <Button size="small" onClick={()=>this.handleSwipe("right")}
             ↪  disabled={clicked <= 0}>
              <KeyboardArrowLeft />
            </Button>
          }
        />
      </div>);
  }

  renderFavorite() {
    const { clicked, liked } = this.state;
    return (<div>
      { liked.hasOwnProperty(clicked) && liked[clicked] ? <FavoriteIcon/> :
       ↪  <FavoriteBorderIcon/> }
    </div>);
  }

  render() {
    const { classes } = this.props;
    const { clicked } = this.state;

    // Handling whether to go back to the Home page or display the Photos page
    if (this.state.exit) {
      console.log("EXITING")
      return <Redirect to={{ pathname: "/Home", state: {page: "home"} }} />
    }

    return (
      <Wrapper isMounted={this.props.isMounted} exit={this.state.exit}>
        <div>
          <div className={classes.container} justify={"center"}>
            <Leap
              photos={this.state.photoRefs}
              handleHover={this.handleHover}
```

```
              handleClick={this.handleClick}
              handleSwipe={this.handleSwipe}
              handleSwipeUp={this.handleSwipeUp}
              handleRotate={this.handleRotate}
              handlePinch={this.handlePinch}
              handleTranslate={this.handleTranslate}
              handleThumb={this.handleThumb}
            />

            {/* Handling whether to render a full screen photo or not */}
            <div className={classes.maincontent}>
              { clicked != -1 ? this.renderFullScreen(clicked) :
               ↪  this.renderPhotos() }
            </div>

            {/* Control bar */}
            <div className={classes.controlbar}>
              {/* Home button: */}
              <div className={classes.control}>
                <Button onClick={() => this.handleExit()}
                 ↪  className={classes.button}>
                  <Home/>
                </Button>
              </div>
              <div className={classes.control}>
                { clicked != -1 ? this.renderFullScreenStepper() :
                 ↪  this.renderStepper() }
              </div>
              <div className={classes.control}>
                { clicked != -1 ? this.renderFavorite() : <div></div> }
              </div>
            </div>
          </div>
        </div>
      </Wrapper>
    );
  }
}

export default withStyles(styles)(PhotosApp);
```

leap.js

```
import PropTypes from 'prop-types';
import React from 'react';
import ReactDOM from 'react-dom';
import { withStyles } from '@material-ui/core/styles';
import GestureHandler from '../../gesture-interface/app-interface' // Added

const fingers = ["#9bcfedBB", "#B2EBF2CC", "#80DEEABB", "#4DD0E1BB", "#26C6DABB"];
const left_fingers = ["#d39bed", "#e1b1f1", "#ca80ea", "#b74ce1", "#a425da"];
```

```
const paused_fingers = ["#9bed9b", "#b1f0b1", "#80ea80", "#4ce14c", "#25da25"];

const styles = {
    canvas: {
        position: 'absolute',
        height: '100%',
        width: '100%',
        zIndex: 10,
        pointerEvents: 'none',
    }

};

class Leap extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            indexFinger: "",
            hovered: -1,
            clicked: -1,
            pause: 4,
        }
    }

    componentDidMount() {
        let gestureHandler = new GestureHandler();
        this.gestureHandler = gestureHandler;
        gestureHandler.onEachGesture(function (gesture) {
            console.log(gesture);
            this.setState({ pause: 4 });
        }.bind(this));
        gestureHandler.onGesture("rhand_lswipe", function () {
            this.props.handleSwipe("left");
        }.bind(this));
        gestureHandler.onGesture("rhand_rswipe", function () {
            this.props.handleSwipe("right");
        }.bind(this));
        gestureHandler.onGesture("rhand_uswipe", function () {
            let clicked = this.state.clicked;
            if (clicked != -1) {
                gestureHandler.removePoseHandler("grab");
                gestureHandler.removePoseHandler("pinch");
                gestureHandler.removePoseHandler("thumb");
                clicked = -1;
                this.setState({ clicked });
            }
            this.props.handleSwipeUp();
        }.bind(this));
        gestureHandler.onGesture("rindex_airtap", function () {
            var { clicked, hovered } = this.state;
            if (clicked == -1 && hovered != -1) {
```

```
                gestureHandler.onPose("grab", function (data) {
                    this.props.handleRotate(data.rotation);
                }.bind(this));
                gestureHandler.onPose("pinch", function (data) {
                    this.props.handlePinch(data.pinch, data.translation);
                }.bind(this));
                gestureHandler.onPose("thumb", function (data) {
                    let thumbUp = data.thumbVector[1] > 40;
                    let thumbDown = data.thumbVector[1] < -40;
                    if (thumbUp) {
                        console.log("THUMB UP");
                        this.props.handleThumb(true);
                    }
                    if (thumbDown) {
                        console.log("THUMB DOWN");
                        this.props.handleThumb(false);
                    }
                }.bind(this));
                clicked = hovered;
                this.props.handleClick(clicked);
                this.setState({ clicked: clicked, hovered: -1});
            }
        }.bind(this));
        this.gestureHandler.onFrame(function (frame) {
            if (frame.fingers.length != 0) {
                this.setState({ hand: true });
            } else {
                this.setState({ hand: false });
            }
            this.traceFingers(frame.fingers);
        }.bind(this));

        this.timer = setInterval(() => {
            if (this.state.pause > 0) {
                this.setState({ pause: this.state.pause - 1 });
            }
            if (this.state.hand) {
                var { clicked, hovered } = this.state;
                if (clicked == -1 && !this.state.amiclicked) {
                    hovered = this.checkHover();
                    this.setState({ hovered });
                    this.props.handleHover(hovered);
                }
            }
        }, 100);
    }

    componentWillUnmount() {
        clearInterval(this.timer);
        this.gestureHandler.disconnect();
    }
```

```
111
112     traceFingers(pointables) {
113         try {
114             // TODO: make canvas and ctx global
115             const canvas = this.refs.canvas;
116             canvas.width = canvas.clientWidth;
117             canvas.height = canvas.clientHeight;
118             const ctx = canvas.getContext("2d");
119             ctx.clearRect(0, 0, canvas.width, canvas.height);
120
121             const { pause } = this.state;
122
123             pointables.forEach((pointable) => {
124                 const color = pause > 0 ? paused_fingers[pointable.type] :
                     ↪  fingers[pointable.type];
125                 const normalized = pointable.normalizedPosition;
126                 const x = ctx.canvas.width * normalized[0];
127                 const y = ctx.canvas.height * (1 - normalized[1]);
128                 const radius = Math.min(20 / Math.abs(pointable.touchDistance),
                     ↪  50);
129                 this.drawCircle([x, y], radius, color, pointable.type === 1);
130
131                 if (pointable.type == 1) {
132                     this.setState({
133                         indexFinger: { x, y, vel: pointable.tipVelocity[2] }
134                     })
135                 }
136             });
137         } catch (err) {
138             // console.log("ERR", err);
139         }
140     }
141
142     drawCircle(center, radius, color, fill) { //draws a circle
143         const canvas = this.refs.canvas;
144         const ctx = canvas.getContext("2d");
145         ctx.beginPath();
146         ctx.arc(center[0], center[1], radius, 0, 2 * Math.PI);
147         ctx.closePath();
148         ctx.lineWidth = 10;
149         if (fill) {
150             ctx.fillStyle = color;
151             ctx.fill();
152         } else {
153             ctx.strokeStyle = color;
154             ctx.stroke();
155         }
156     }
157
158     checkHover() { //returns which photo is hovered
159         const photos = this.props.photos;
```

```
        const { x, y } = this.state.indexFinger;
        for (let i = 0; i < photos.length; i++) {
            if (photos[i] && photos[i].current) {
                const dims = photos[i].current.getBoundingClientRect();
                if (x > dims.left && x < dims.right &&
                    y > dims.top && y < dims.bottom) {
                    return i;
                }
            }
        }
        return -1;
    }

    render() {
        const { classes } = this.props;
        return (
            <canvas className={classes.canvas} ref="canvas"></canvas>
        )
    }

}


Leap.propTypes = {
    photos: PropTypes.array,
    handleHover: PropTypes.func,
    handleSwipe: PropTypes.func,
    handleExit: PropTypes.func,
    handleRotate: PropTypes.func,
    handlePinch: PropTypes.func,
    handleTranslate: PropTypes.func,
    handleThumb: PropTypes.func
};

// TODO: better default values
Leap.defaultProps = {
    photos: [],
};

export default withStyles(styles)(Leap);
```

F.8.5 PDFs App

index.js

```
import React, { Component } from 'react';
import { Redirect } from 'react-router';
import PropTypes from 'prop-types';
import { ButtonBase, Typography } from '@material-ui/core';
import { withStyles } from '@material-ui/core/styles';
```

```
import classNames from 'classnames';
import PDFsApp from './PDFsApp.jsx';
import { Link } from 'react-router-dom'


const styles = theme => ({
    root: {
        display: 'flex',
        flexWrap: 'wrap',
        width: '100%',
        height: '100%'
    },
    image: {
        position: 'relative',
        height: '100%',
        [theme.breakpoints.down('xs')]: {
            width: '100% !important', // Overrides inline-style
        },
    },
    hovered: {
        zIndex: 5,
        '& $imageBackdrop': {
            opacity: 0.15,
        },
        '& $imageMarked': {
            opacity: 0,
        },
        '& $imageTitle': {
            border: '4px solid currentColor',
        },
    },
    imageButton: {
        position: 'absolute',
        left: 0,
        right: 0,
        top: 0,
        bottom: 0,
        display: 'flex',
        alignItems: 'center',
        justifyContent: 'center',
        color: theme.palette.common.white,
    },
    imageSrc: {
        position: 'absolute',
        left: 0,
        right: 0,
        top: 0,
        bottom: 0,
        backgroundSize: 'cover',
        backgroundPosition: 'center 40%',
    },
```

```
    imageBackdrop: {
        position: 'absolute',
        left: 0,
        right: 0,
        top: 0,
        bottom: 0,
        backgroundColor: theme.palette.common.black,
        opacity: 0.4,
        transition: theme.transitions.create('opacity'),
    },
    imageTitle: {
        position: 'relative',
        padding: `${theme.spacing.unit * 2}px ${theme.spacing.unit * 4}px
        ↪  ${theme.spacing.unit + 6}px`,
    },
    imageMarked: {
        height: 3,
        width: 18,
        backgroundColor: theme.palette.common.white,
        position: 'absolute',
        bottom: -2,
        left: 'calc(50% - 9px)',
        transition: theme.transitions.create('opacity'),
    },
});

const image = {
    url:
    ↪  'https://images.unsplash.com/photo-1523906891494-64b456bf3c9e?ixlib=rb-1.2.1&ixid=eyJ
    title: 'PDFs',
    width: '100%',
};

class PDFs extends Component {

    constructor(props) {
        super(props);
    }

    render() {
        const { classes, hovered, clicked } = this.props;

        if (clicked) {
            return <Redirect to={{pathname: "/PDFs"}} />
        } else {
            // icon
            return (
                <ButtonBase
                    focusRipple
                    key={image.title}
                    className={hovered? classNames(classes.image, classes.hovered)
                    ↪  : classes.image}
```

```
                    focusVisibleClassName={classes.focusVisible}
                    style={{
                        width: image.width,
                    }}
                    onClick  = {this.props.onclick}
                >
                    <span
                        className={classes.imageSrc}
                        style={{
                            backgroundImage: `url(${image.url})`,
                        }}
                    />
                    <span className={classes.imageBackdrop} />
                    <span className={classes.imageButton}>
                        <Typography
                            component="span"
                            variant="subheading"
                            color="inherit"
                            className={classes.imageTitle}
                        >
                            {image.title}
                            <span className={classes.imageMarked} />
                        </Typography>
                    </span>
                </ButtonBase>
            );
        }
    }
}

PDFs.propTypes = {
    hovered: PropTypes.bool,
    clicked: PropTypes.bool,
};

export default withStyles(styles)(PDFs);
```

PDFsApp.jsx

```
import React, { Component } from 'react';
import { Redirect } from 'react-router';
import { withStyles } from '@material-ui/core/styles';
import glamorous from 'glamorous'
import classNames from 'classnames';
import Leap from './leap.js';
import { Grid } from '@material-ui/core';
import SwipeableViews from 'react-swipeable-views';
import MobileStepper from '@material-ui/core/MobileStepper';
import Button from '@material-ui/core/Button';
import Pagination from '@material-ui/lab/Pagination';
import KeyboardArrowLeft from '@material-ui/icons/KeyboardArrowLeft';
```

```
import KeyboardArrowRight from '@material-ui/icons/KeyboardArrowRight';
import Home from '@material-ui/icons/Home';
import Clear from '@material-ui/icons/Clear';

import { css } from 'glamor';
import { Transition } from 'react-transition-group';
import Carousel from 'react-responsive-carousel';
//firebase
import * as firebase from "firebase/app";
import "firebase/database";

import { Document, Page, pdfjs } from "react-pdf";
pdfjs.GlobalWorkerOptions.workerSrc =
 ↪  `//cdnjs.cloudflare.com/ajax/libs/pdf.js/${pdfjs.version}/pdf.worker.js`;

const zoomIn = css.keyframes({
  '0%': { transform: 'scale(0.5)' },
  '100%': { transform: 'scale(1)' }
})

//CSS:
const styles = {

  gallery: {
    animation: `${zoomIn} 1s`
  },

  container: {
    display: 'flex',
    flexDirection: 'column',
    position: 'absolute',
    top: '0px',
    left: '0px',
    width: '100%',
    height: '100%',
    padding: '0px',
    listStyle: 'none',
    overflow: 'hidden',
    zIndex: '1',
    backgroundColor: '#CFD8DC',
  },

  carousel: {
    // width: '90%',
    maxHeight: '95%',
    padding: '0px',
    margin: '0px',
    overflow: 'hidden',
  },

  row: {
```

```
    maxHeight: '50vh',
  },

  cell: {
    display: 'inline-block',
    maxWidth: '85%',
    maxHeight: '85%',
    verticalAlign: 'middle',
    boxSizing: 'border-box',
    margin: '0px',
    padding: '3%',
    position: 'relative',
  },

  thumbnail: {
    display: 'block',
    maxWidth: '90%',
    maxHeight: '60%',
    width: 'auto',
    height: '300px',
    margin: 'auto',
    padding: '3%',
    border: 'none',
    transform: 'scale(1)',
    transition: 'all 0.4s',
    boxShadow: '0px 0px 10px 2px #999',
    backgroundColor: "#ECEFF1",
    position: "relative",
    zIndex: '1'
  },

  hovered: {
    transform: 'scale(1.5)',
    animationDuration: '0.1s',
    zIndex: '15 !important',
    cursor: 'pointer',
  },

  dots: {
    margin: 'auto',
  },

  maincontent: {
    position: 'absolute',
    bottom: '50px',
    top: '50px',
    left: 0,
    right: 0,
    overflow: 'hidden'
  },

```

```
  pdf: {
    position: 'relative',
    height: 'fit-content',
    width: 'fit-content',
    margin: 'auto'
  },

  controlbar: {
    boxSizing: 'border-box',
    display: 'flex',
    justifyContent: 'space-between',
    alignItems: 'center',
    margin: 0,
    padding: '5px',
    position: 'absolute',
    bottom: 0,
    left: 0,
    right: 0,
    height: '50px'
  },

  control: {
    flex: '1 1 0'
  },

  pagination: {
    margin: 'auto',
    width: 'fit-content',
    maxWidth: '100%'
  },

  stepper: {
    margin: 'auto',
    width: 'fit-content',
    maxWidth: '100%',
    backgroundColor: 'rgb(0,0,0,0)'
  },

  button: {
    position: 'relative',
    height: '100%',
    color: "rgba(50,50,50,0.8)"
  },

  xbutton: {
    position: 'fixed',
    top: '10px',
    right: '10px',
    color: "rgba(50,50,50,0.8)"
  }
};
```

```
165
166 const fadeIn = css.keyframes({
167   '0%': { opacity: 0 },
168   '100%': { opacity: 1 }
169 })
170 const slideOut = css.keyframes({
171   '100%': { transform: 'translateY(-100%)' },
172 })
173
174 //Animations entering/exiting this page:
175 const Wrapper = glamorous.div(props => ({
176   animation: props.isMounted ? `${slideOut} 2.5s` : `${fadeIn} 1.5s`,
177   position: 'absolute',
178   top: '0px',
179   left: '0px',
180   width: '100vw',
181   height: '100vh',
182   zIndex: 5
183 }))
184 const pdfs = [require('./pdfs/test1.pdf'),
185               require('./pdfs/test2.pdf'),
186               require('./pdfs/test3.pdf'),
187               require('./pdfs/test1.pdf'),
188               require('./pdfs/test2.pdf'),
189               require('./pdfs/test3.pdf'),
190               require('./pdfs/test1.pdf'),
191               require('./pdfs/test2.pdf'),
192               require('./pdfs/test3.pdf')]
193
194 //firebase
195 const firebaseConfig = {
196   apiKey: "AIzaSyDjM37_DSv2RvPQzl5YiVzmgRHfpd4rJFU",
197   authDomain: "lui-medialab.firebaseapp.com",
198   databaseURL: "https://lui-medialab.firebaseio.com",
199   projectId: "lui-medialab",
200   storageBucket: "lui-medialab.appspot.com",
201   messagingSenderId: "247289397118",
202   appId: "1:247289397118:web:eb2bcb0076d4bb4d"
203 };
204
205 if (!firebase.apps.length) {
206 firebase.initializeApp(firebaseConfig);
207 }
208 var database = firebase.database();
209 var currentRef = database.ref('voice');
210 //end
211 class PDFsApp extends Component {
212   constructor(props) {
213     super(props);
214
215     let refs = [];
```

```
    for (let i = 0; i < pdfs.length; i++) {
      refs.push(React.createRef());
    }

    this.state = {
      refs: refs,
      numPreviewPages: Math.ceil(refs.length / 8),
      numPages: null,
      currentPage: 1,
      hovered: -1,
      clicked: -1,
      index: 0,
      exit: false,
      amiclicked:false,
      rotation: 0, // Rotate pdf
      zoom: 1.0
    };

    this.handleClick = this.handleClick.bind(this);
  }

  componentDidMount() {
    //google home
    currentRef.update({"current":"pdfs"});
    var something = this;
    currentRef.on('value', function(snapshot) {
      var db = snapshot.val();
      var name = db.goto;
      if (db.update){
        if (name === "home") {
            something.setState({ exit: true });
            currentRef.update({"update":false});
        }

      }
      if(db.clicked){
        currentRef.update({"clicked":false});
        something.gestureDetected = true;
        something.handleClick(db.hovered);
      }
      if (db.back){
        if (something.state.amiclicked){
          something.handleSwipeUp();
        }
        else{
          something.setState({ exit: true });

        }
        currentRef.update({"back":false});
      }
    });
```

```
  }

  rotate(){
    let newRotation = this.state.rotation + 90;
    if(newRotation >= 360){
      newRotation =- 360;
    }
    this.setState({
      rotation: newRotation,
    })
  }

  handleHover = (pdf) => {
    this.setState({ hovered: pdf })
    currentRef.update({"hovered": pdf});
  }

  handleClick = (pdfId) => {
    this.setState({ clicked: pdfId })
    this.setState({ amiclicked: true })
    this.setState({ currentPage: 1 })
    this.setState({ rotation: 0 })
  }

  handleExit = () => {
    this.setState({
      exit: true
    })
  }

  handleRotate = (handRotation) => {
    this.setState(prevState => ({ rotation: (prevState.rotation + 180 *
     ↪  handRotation / 3.14) }))
  }

  handlePinch = (pinch) => {
    let { clicked, zoom } = this.state;
    if (clicked != -1) {
      zoom *= ((pinch - 1) * 1.2) + 1;
      if (zoom < 0.4) {
        zoom = 0.4;
      } else if (zoom > 4) {
        zoom = 4;
      }
      this.setState({ zoom })
    }
  }

  handleSwipe = (dir) => {
    let { clicked, numPages, currentPage, index, numPreviewPages } = this.state;
    if (dir === "left") {
```

```
      console.log("swipe left");
      if (clicked != -1) {
        this.setState({ currentPage: (currentPage < numPages) ? currentPage + 1 :
        ↪  currentPage })
      } else if (index < numPreviewPages - 1) {
        this.setState({ index: index + 1 })
      }
    } else {
      console.log("swipe right");
      if (clicked != -1) {
        this.setState({ currentPage: (currentPage > 1) ? currentPage - 1 :
        ↪  currentPage })
      } else if (index > 0) {
        this.setState({ index: index -1 })
      }
    }
  }

  handleNext = () => {
    let { index, numPreviewPages } = this.state;
    if (index < numPreviewPages - 1) {
      this.setState({ index: index + 1 });
    }
  };

  handleBack = () => {
    let { index } = this.state;
    if (index > 0) {
      this.setState({ index: index - 1 });
    }
  };

  handleSwipeUp = () => {
    let { clicked } = this.state;
    if (clicked != -1) {
      this.setState({ clicked: -1 });
      this.setState({ amiclicked: false });
    } else {
      this.setState({ exit: true });
    }
  }

  renderPDF(index) { //renders a pdf in the grid
    const { classes } = this.props;
    const { hovered, refs } = this.state;
    const ref = refs[index]

    return (<Grid item className={classes.cell} ref={ref} xs={12} sm={3}>
      <div
        onMouseEnter={() => { this.setState({hovered: index}) }}
        onMouseLeave={() => { this.setState({hovered: -1}) }}
```

```
366         onClick={() => { this.handleClick(index) }}
367         className={hovered === index ? classNames(classes.thumbnail,
            ↪  classes.hovered) : classes.thumbnail} >
368           {pdfs[index]}
369       </div>
370     </Grid>);
371   }
372
373   onDocumentLoadSuccess = (document) => {
374     console.log("loaded")
375     const { numPages } = document;
376     this.setState({
377       numPages,
378       pageNumber: 1,
379     });
380   };
381
382   goToPage(page) {
383     const { numPages } = this.state;
384     if (page > 0 && page <= numPages) {
385       this.setState({ currentPage: page })
386     }
387   }
388
389   renderFullScreen(index) { //renders the full screen gallery view for the pdfs
390     const { classes } = this.props;
391     const { rotation, zoom, currentPage, numPages } = this.state;
392     let pdfRotation = Math.round(rotation / 90) * 90;
393     return (<div>
394       {/* PDF */}
395         <div className={classes.pdf} justify={"center"}>
396             <Document
397               file={pdfs[index]}
398               onLoadSuccess={this.onDocumentLoadSuccess}
399               onLoadError={console.error}
400               rotate={pdfRotation}
401             >
402               <Page pageNumber={currentPage} scale={zoom}
                  ↪  className={classes.pdfpage} />
403             </Document>
404         </div>
405       {/* Exit button: */}
406       <Button onClick={() => this.handleSwipeUp()}  className={classes.xbutton}>
407         <Clear/>
408       </Button>
409     </div>);
410   }
411
412   renderPDFs() { //renders the full grid of pdfs
413     const { classes } = this.props;
414     const { index, refs } = this.state;
```

```
    let pages = [];
    let rows = [];
    let previews = [];
    var i;
    for (i = 0; i < refs.length; i++) {
      previews.push(this.renderPDF(i));
      if ((i + 1) % 4 == 0) {
        rows.push(<Grid container className={classes.row} spacing={0}
        ↪  justify={"center"} >
          {previews}
        </Grid>);
        previews = [];
      }
      if ((i + 1) % 8 == 0) {
        pages.push(<div className={classes.carousel} justify={"center"}>
          {rows}
        </div>);
        rows = [];
      }
    }
    if ((refs.length % 8) != 0) {
      if ((refs.length % 4) != 0) {
        rows.push(<Grid container className={classes.row} spacing={0}
        ↪  justify={"center"} >
          {previews}
        </Grid>);
      }
      pages.push(<div className={classes.carousel} justify={"center"}>
        {rows}
      </div>);
    }

    return (<div>
      <div>
        <SwipeableViews className={classes.gallery} index={index}>
          {pages}
        </SwipeableViews>
      </div>
    </div>);
  }

  renderStepper() {
    const { classes } = this.props;
    const { index, numPreviewPages } = this.state;

    return (numPreviewPages <= 1) ? "" : (<div className = "stepper">
      <MobileStepper
        variant="dots"
        steps={numPreviewPages}
        position="bottom"
        activeStep={index}
```

```
        className={classes.stepper}
        classes={{ dots: classes.dots }}
        nextButton={
          <Button size="small" onClick={this.handleNext} disabled={index >=
           ↪  numPreviewPages - 1}>
            <KeyboardArrowRight />
          </Button>
        }
        backButton={
          <Button size="small" onClick={this.handleBack} disabled={index <= 0}>
            <KeyboardArrowLeft />
          </Button>
        }
      />
    </div>);
  }

  renderPageSelection() {
    const { classes } = this.props;
    const { numPages, currentPage } = this.state;
    return <div className = "pagination">
        {/* Page number at the bottom of the view: */}
        <Pagination className={classes.pagination} count={numPages}
         ↪  page={currentPage} onChange={(event, page) => this.goToPage(page)}
         ↪  shape="rounded" />
      </div>
  }

  render() {
    const { classes } = this.props;
    const { clicked } = this.state;

    // Handling whether to go back to the Home page or display the PDFs page
    if (this.state.exit) {
      console.log("EXITING")
      return <Redirect to={{ pathname: "/Home", state: {page: "home"} }} />
    }

    return (
      <Wrapper isMounted={this.props.isMounted} exit={this.state.exit}>
        <div>
          <div className={classes.container} justify={"center"}>
            <Leap
              pdfs={this.state.refs}
              handleHover={this.handleHover}
              handleClick={this.handleClick}
              amiclicked = {this.state.amiclicked}
              handleSwipe={this.handleSwipe}
              handleSwipeUp={this.handleSwipeUp}
              handleRotate={this.handleRotate}
              handlePinch={this.handlePinch}
```

```
            />

            {/* Handling whether to render a full screen pdf or not */}
            <div className={classes.maincontent}>
              { clicked != -1 ? this.renderFullScreen(clicked) : this.renderPDFs()
               ↪ }
            </div>

            {/* Control bar */}
            <div className={classes.controlbar}>
              {/* Home button: */}
              <div className={classes.control}>
                <Button onClick={() => this.handleExit()}
                 ↪  className={classes.button}>
                  <Home/>
                </Button>
              </div>
              <div className={classes.control}>
                { clicked != -1 ? this.renderPageSelection() : this.renderStepper()
                 ↪ }
              </div>
              <div className={classes.control}>
              </div>
            </div>
          </div>
        </div>
      </Wrapper>
    );
  }
}

export default withStyles(styles)(PDFsApp);
```

leap.js

```
import PropTypes from 'prop-types';
import React from 'react';
import ReactDOM from 'react-dom';
import { withStyles } from '@material-ui/core/styles';
import Home from '@material-ui/icons/Home';
import GestureHandler from '../../gesture-interface/app-interface';

const fingers = ["#9bcfedBB", "#B2EBF2CC", "#80DEEABB", "#4DD0E1BB", "#26C6DABB"];
const paused_fingers = ["#9bed9b", "#b1f0b1", "#80ea80", "#4ce14c", "#25da25"];

const styles = {
    canvas: {
        position: 'absolute',
        height: '100%',
        width: '100%',
        zIndex: 10,
```

```
        pointerEvents: 'none',
    }
};

class Leap extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            hand: false,
            indexFinger: "",
            hovered: -1,
            clicked: -1,
            amiclicked: "",
            pause: 4,
            lhovered: "",
            intervalId1: 0,
            intervalId2: 0,
            intervalId3: 0,
            intervalId4: 0,
        }
    }
    componentWillReceiveProps(nextProps) {
        this.setState({ amiclicked: nextProps.amiclicked });
    }

    componentDidMount() {
        let gestureHandler = new GestureHandler();
        this.gestureHandler = gestureHandler;
        gestureHandler.onEachGesture(function (gesture) {
            console.log(gesture);
            this.setState({ pause: 4 });
        }.bind(this));

        gestureHandler.onGesture("rhand_lswipe", function () {
            this.props.handleSwipe("left");
        }.bind(this));

        gestureHandler.onGesture("rhand_rswipe", function () {
            this.props.handleSwipe("right");
        }.bind(this));

        gestureHandler.onGesture("rhand_uswipe", function () {
            let clicked = this.state.clicked;
            if (clicked != -1) {
                gestureHandler.removePoseHandler("grab");
                gestureHandler.removePoseHandler("pinch");
                clicked = -1;
                this.setState({ clicked });

            }
            this.props.handleSwipeUp();
```

```
68          }.bind(this));
69
70          gestureHandler.onGesture("rindex_airtap", function () {
71              var hovered = this.state.hovered;
72              if (hovered != -1) {
73                  gestureHandler.onPose("grab", function (data) {
74                      this.props.handleRotate(data.rotation);
75                  }.bind(this));
76                  gestureHandler.onPose("pinch", function (data) {
77                      this.props.handlePinch(data.pinch);
78                  }.bind(this));
79                  const clicked = hovered;
80                  this.props.handleClick(clicked);
81                  this.setState({ clicked: clicked, hovered: -1});
82              }
83          }.bind(this));
84
85          gestureHandler.onFrame(function (frame) {
86              if (frame.fingers.length != 0) {
87                  this.setState({ hand: true });
88              } else {
89                  this.setState({ hand: false });
90              }
91              this.traceFingers(frame.fingers);
92          }.bind(this));
93
94          this.timer = setInterval(() => {
95              if (this.state.pause > 0) {
96                  this.setState({ pause: this.state.pause - 1 });
97              }
98              if (this.state.hand) {
99                  var { clicked, hovered } = this.state;
100                 if (clicked == -1 && !this.state.amiclicked) {
101                     hovered = this.checkHover();
102                     this.setState({ hovered });
103                     this.props.handleHover(hovered);
104                 }
105             }
106
107         }, 100);
108     }
109
110     componentWillUnmount() {
111         clearInterval(this.timer);
112         this.gestureHandler.disconnect();
113     }
114
115     traceFingers(pointables) {
116         try {
117             // TODO: make canvas and ctx global
118             const canvas = this.refs.canvas;
```

```
            canvas.width = canvas.clientWidth;
            canvas.height = canvas.clientHeight;
            const ctx = canvas.getContext("2d");
            ctx.clearRect(0, 0, canvas.width, canvas.height);

            const { pause } = this.state;

            pointables.forEach((pointable) => {
                const color = pause > 0 ? paused_fingers[pointable.type] :
                 ↪  fingers[pointable.type];
                const normalized = pointable.normalizedPosition;
                const x = ctx.canvas.width * normalized[0];
                const y = ctx.canvas.height * (1 - normalized[1]);
                const radius = Math.min(20 / Math.abs(pointable.touchDistance),
                 ↪  50);
                this.drawCircle([x, y], radius, color, pointable.type === 1);

                if (pointable.type == 1) {
                    this.setState({
                        indexFinger: { x, y, vel: pointable.tipVelocity[2] }
                    })
                }
            });
        } catch (err) {
            // console.log("ERR", err);
        }
    }

    darken() {
        const canvas = this.refs.canvas;
        const ctx = canvas.getContext("2d");
        ctx.beginPath();
        ctx.rect(20, 20, 150, 100);
        ctx.fillStyle = "red";
        ctx.fill();
    }

    getCoords(x, y, degrees, radius) {
        /** Calculates the coordinates for the center of a button relative
         * to the center of the radial menu.
         *
         * params: x,y are the center of the radial menu
         * degrees is the angle of the button from the horizontal
         * radius is the distance from the center of the radial menu to the center
↪   of the button
         *
         * returns: coordinates of the center of the button
         */
        const newx = x + radius * Math.cos(degrees * Math.PI / 180)
        const newy = y + radius * Math.sin(degrees * Math.PI / 180)
        return [newx, newy]
```

```
    }

    clamp(val, min, max) {
        return Math.min(Math.max(val, min), max);
    }

    rollToVolume(roll) {
        return this.clamp(2 - roll, 0, 3) * 100 / 3;
    }

    drawLabel(center) { //label for button
        const canvas = this.refs.canvas;
        const ctx = canvas.getContext("2d");
        ctx.font = "40px Helvetica";
        // const coords = this.getCoords(x, y, )
        ctx.fillText("Home", center[0] - 50, center[1] + 15);
    }

    drawArc(center, radius, color, percent) {
        const canvas = this.refs.canvas;
        const ctx = canvas.getContext("2d");
        ctx.beginPath();
        ctx.arc(center[0], center[1], radius, 0, percent * 2 * Math.PI, false);
        ctx.lineWidth = 10;
        ctx.strokeStyle = color;
        ctx.stroke();
    }

    drawCircle(center, radius, color, fill) { //draws a circle
        const canvas = this.refs.canvas;
        const ctx = canvas.getContext("2d");
        ctx.beginPath();
        ctx.arc(center[0], center[1], radius, 0, 2 * Math.PI);
        ctx.closePath();
        ctx.lineWidth = 10;
        if (fill) {
            ctx.fillStyle = color;
            ctx.fill();
        } else {
            ctx.strokeStyle = color;
            ctx.stroke();
        }
    }

    drawPointer(center, radius, color, fill) { //draw small circle that points to
     ↪  what is hovered
        const canvas = this.refs.canvas;
        const ctx = canvas.getContext("2d");
        ctx.beginPath();
        ctx.arc(center[0], center[1], radius, 0, 2 * Math.PI, false);
        ctx.closePath();
```

```
        ctx.lineWidth = 10;
        if (fill) {
            ctx.fillStyle = color;
            ctx.fill();
        } else {
            ctx.strokeStyle = color;
            ctx.stroke();
        }
    }

    checkHover() { //returns which pdf is hovered
        const pdfs = this.props.pdfs;
        const { x, y } = this.state.indexFinger;
        for (let i = 0; i < pdfs.length; i++) {
            if (pdfs[i] && pdfs[i].current) {
                const dims = pdfs[i].current.getBoundingClientRect();
                if (x > dims.left && x < dims.right &&
                    y > dims.top && y < dims.bottom) {
                    return i;
                }
            }
        }
        return (-1);
    }

    render() {
        const { classes } = this.props;
        return (
            <canvas className={classes.canvas} ref="canvas"></canvas>
        )
    }
}

Leap.propTypes = {
    pdfs: PropTypes.array,
    handleHover: PropTypes.func,
    handleSwipe: PropTypes.func,
    handleExit: PropTypes.func,
    handleRotate: PropTypes.func,
    handlePinch: PropTypes.func
};

// TODO: better default values
Leap.defaultProps = {
    pdfs: [],
};

export default withStyles(styles)(Leap);
```

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**